Tecnológico de Costa Rica

Compiladores e Intérpretes

Grupo 40

Profesor: Dr. Francisco Torres Rojas

Apuntes de clase: 19 de mayo de 2017

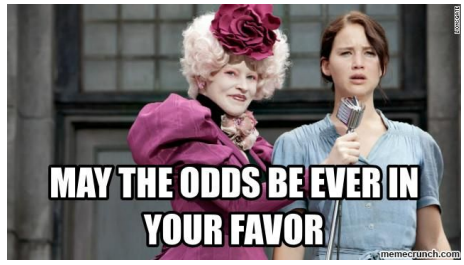
Apuntador: Víctor Andrés Chaves Díaz

Carné: 2015107095

2017

Tabla de contenidos

Notas Pendientes



- Solo nos quedan 2 clases más (miércoles 24 y viernes 26).
- En esas dos clases, las personas que NO hayan sido apuntadores se tienen que pelear en una batalla a la muerte los dos campos de apuntadores (ya en serio, lleguen con tiempo para ser apuntadores).
- El proyecto se entrega el 31 de mayo
- El segundo examen es el 2 de junio, y el 9 es el examen final para los que no se hayan eximido.
- Recuerden que el lapicero aquel aún sigue siendo una opción válida para pasar el curso, consultar con los apuntes anteriores para más información.

Análisis sintáctico - Tabla de Parsing

Hasta ahora, las tablas que hemos visto se nos dieron por un ente mágico, que puede ser o no malvado, pero ahora ya sabemos como hacerlas nosotros mismos!

- Construcción de Tabla de Parsing
  Debemos crear una tabla M, que debe tener:
  - Una fila por cada no terminal
  - Una columna por cada terminal y $
- Se hace el cálculo de `FIRST` y de `FOLLOW`, ***EN ESE ORDEN Y SIN BRINCAR PASOS***.
- Posterior a esto, se calcula el `PREDICT` de cada regla
- Para toda regla de la forma A → α
- Para cada elemento a en PREDICT(A → α), agregue la regla A → α en la casilla M[A][a]

Tan tan! Sencillito, hasta ~~Charlton~~ ~~Heston~~ Moisés hace muchos cientos de años pudo, con la primera tabla de parsing, que hasta hizo otra más.

Ejemplos de tabla de parsing

Ejemplo 1

Regla	PREDICT
1) S → (S)S	{(}
2) S → ε	{), $}

Tabla de Parsing:

	(	)	$
S	1	2	2

Ejemplo 2

Regla	PREDICT
1) E → T E’	**{(, #}**
2) E’ → OP T E’	**{+ , -}**
3) E’ → ε	**{$,)}**
4) OP → +	**{+}**
5) OP → -	**{-}**
6) T → F T’	**{(, #}**
7) T’ → M F T’	**{*}**
8) T’ → ε	**{$,), +, -}**
9) M → *	**{*}**
10. F → (E)	**{(}**
11. F → #	**{#}**

Tabla de Parsing:

	(	#	)	+	-	*	$
E	1	1					
E’			3	2	2		3
OP				4	5		
T	6	6					
T’			8	8	8	7	8
M						9	
F	10	11					

Ejemplo 3 (una secuencia de instrucciones)

No Terminal	FIRST		
	Inicial	Pasada 1	Pasada 2
SQ	∅	∅	{<statement>}
S	∅	{<statement>}	{<statement>}
SQ'	∅	{;,ε}	{;,ε}

No Terminal	FOLLOW	
	Inicial	Pasada 1
SQ	{$}	{$}
S	∅	{;,$}
SQ'	∅	{$}

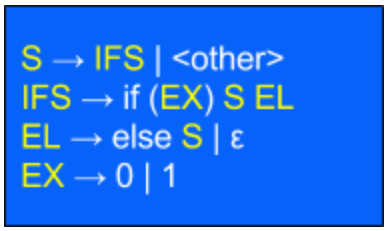
Regla	FIRST(α)	PREDICT
1) SQ → S SQ'	{<statement>}	{<statement>}
2) SQ' → SQ	{;}	{;}
3) SQ' → ε	{ε}	{$}
4) S → <statement>	{<statement>}	{<statement>}

Tabla de Parsing:

	<statement>	;	$
SQ	1		
S	4		
SQ’		2	3

Ejemplo 4

Ahora, hagamos un ejemplo que no hayamos visto: if-then-else



Comenzamos con la tabla de `FIRST`

	FIRST		
No Terminal	Inicial	Pasada 1	Pasada 2
S	∅	{<other>}	{if, <other>}
IFS	∅	{if}	{if}
EL	∅	{else, ε}	{else, ε}
EX	∅	{0, 1}	{0, 1}

Seguimos con la tabla de FOLLOW

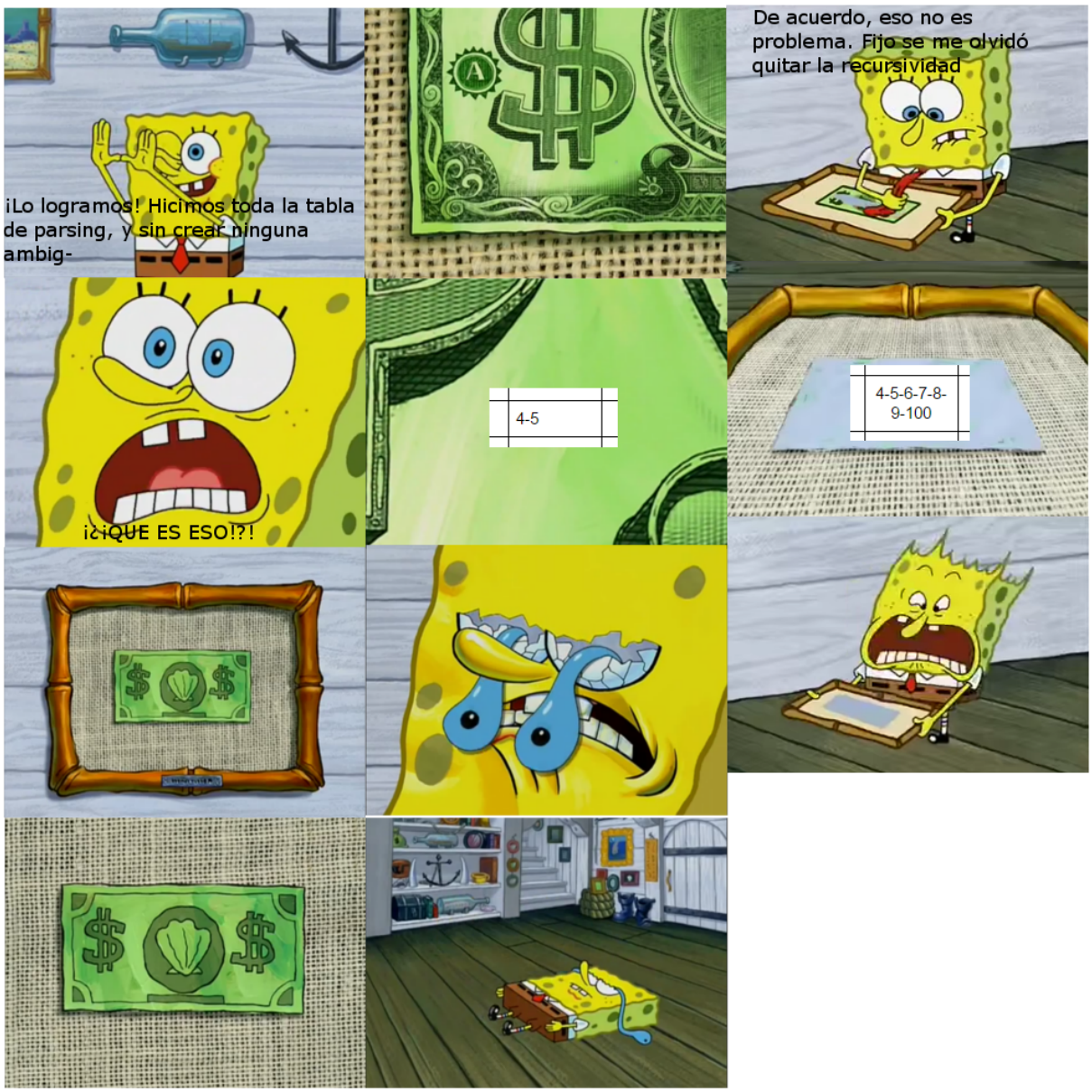
	FOLLOW		
No Terminal	Inicial	Pasada 1	Pasada 2
S	{$}	{$, else}	{$, else}
IFS	∅	{$}	{$, else}
EL	∅	{$}	{$, else}
EX	∅	{)}	{)}

Seguimos con la tabla de PREDICT

Regla	FIRST(α)	PREDICT
1) S → IFS	{if}	{if}
2) S → <other>	{<other>}	{<other>}
3) IFS → if (EX) S EL	{if}	{if}
4) EL → else S	{else}	{else}
5) EL → ε	{ε}	{$, else}
6) EX → 0	{0}	{0}
7) EX → 1	{1}	{1}

Y finalmente construimos la tabla de Parsing

	if	<other>	else	0	1	$
S	1	2				
IFS	3					
EL			4-5			5
EX				6	7	



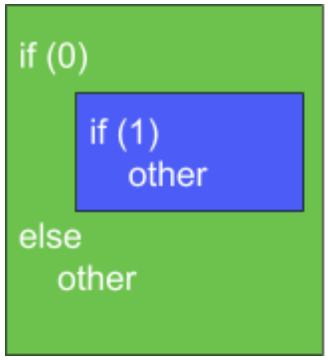
No, damas y caballeros (y otras denominaciones formales cuyos nombres ignoro), no está leyendo mal esta tabla, en efecto *hay un conflicto de reglas*, y esto forma una ambigüedad **grave**. La pregunta del millón es: ¿qué demonios hacemos?

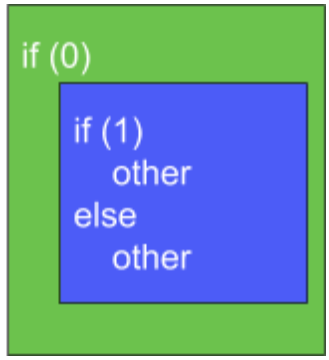
Ambigüedad del if-then-else

- Para demostrar el problema, considere el siguiente código

```
if (0) if (1) other else other
```

- Con la gramática actual, existen dos maneras de poder interpretar este error





- La duda que queda es, ¿cúal es correcta? Ciertamente ambas son válidas, pero preferimos la del lado derecho por ser más natural.
- Se resuelve haciendo un macheteo profesional a la gramática: se fuerza a que el `else` se asocie con el `if` abierto más cercano.
    - Es así como funciona en el lenguaje Izcar, así llamado como su creador, Izcar.
    - Nota: las reglas de la gramática ***NO*** se cambian ni se borran, porque cambian el lenguaje y entonces ahí si se pasea uno en el lenguaje.
- Para hacer este asocie, se hace un cambio a la tabla de parsing, donde se utiliza la regla 4 y no la 5.
- Este problema se conoce como el problema del *dangling else*.

Eliminando ambigüedad del if-then-else

	if	<other>	else	0	1	$
S	1	2				
IFS	3					
EL			4			5
EX				6	7	

Y con esto solucionamos el problema :D

Resumen

- Se toma una CFG.
- Se elimina la recursividad por la izquierda.
- Se factoriza por la izquierda.
- Calcular `FIRST` para todos los no terminales.
- Calcular `FOLLOW` para todos los no terminales.
- Calcular `PREDICT` para todas las reglas.
- Construir la tabla de parsing.

Debe hacerse nota de que *no todas las gramáticas permiten hacer este proceso*.

Análisis Sintáctico - Parsing LL(1)

Parsing LL(1)

- Tipo de parsing Top Down.
- Predictivo, no se hace backtracking.
- Llamado LL(1) por:
    - L → “Left to Right”, la hilera se procesa de izquierda a derecha.
    - L → “Leftmost”, el parser genera una derivación leftmost.
    - 1 → Se utiliza un lookahead de 1 token

LL(1) PARSING TABLE

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

Rashmi K C Dept Of CSE 33

- Fueron estudiados en 1968 por Richard Edwin Stearns y Philip M. Lewis, en aparentemente un artículo llamado *Syntax-Directed Transduction*.
- Es el tipo de parsing que hemos estado viendo en clases! :D

Gramáticas LL(1)

- Definición:
    - Una CFG es LL(1) si para cualquier par de reglas diferentes $A \rightarrow \alpha$ y $A \rightarrow \beta$ se cumple que $\texttt{PREDICT}(A \rightarrow \alpha) \cap \texttt{PREDICT}(A \rightarrow \beta) = \varnothing$
- Una gramática es LL(1) si se puede construir una tabla de parsing LL(1) para ella.
- Las gramáticas LL(1) son un subconjunto *propio* de CFG.
    - Esto nos indica que no todas las CFG son LL(1).
    - Pensemos en el lenguaje natural, que NO es un LL(1).

Nota: GNU Bison no es un parser LL(1), Bison trabaja con LALR(1), que es más poderoso que LL(1), y es suficientemente inteligente que se traga problemas como el dangling else o shift-reduce.

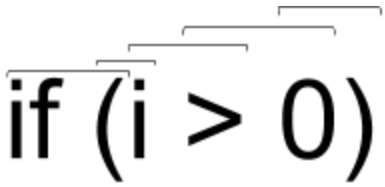
Propiedades de Parsers LL(1)

Sea G un GFC, si G es LL(1), entonces:

1. Si una hilera no se rechaza, está garantizado que vamos a obtener la derivación leftmost correcta que la genera
2. G no es ambigua
3. La complejidad en tiempo y espacio del parsing LL(1) es lineal, con respecto al largo de la hilera a procesar (es decir, es O(n)).

Nota: Nuestro gran amigo Backus no tenía acceso a esta información, hay recordar que FORTRAN se desarrolló en los 50s, las investigaciones de parsing por Stearns y Lewis no se hicieron hasta 1968. Backus tuvo que machetear como si estuviera en una jungla su pobre compilador de FORTRAN. Hay que agradecer a los compiladores cada vez que se utilizan.

Parsers LL(2)



Cada paréntesis cuadrado corresponde a una pareja de tokens en un LL(2)

- Como ya se mencionó, LL(1) trabaja con solo un token de lookahead.
- LL(1) no puede parsear todas las gramáticas, como fue el ejemplo del if-else (técnicamente no es LL(1), le metimos mano).
- Y, ¿si tuviésemos un lookahead de DOS tokens?
- Surgen así los parsers LL(2).
- Se tienen que definir las funciones:
    - $FIRST_2()$
    - $FOLLOW_2()$
    - $PREDICT_2()$
- Cálculos son más caros, se tienen que hacer más, y por ende se ocupa más espacio.
- La tabla de LL(2) tiene $n^2 + n$ columnas, con n terminales en la gramática.

Tabla de Parsing LL(2)

Revisemos una gramática ya vista, ahora con LL(2)

1. S → (S)S
2. S → ε

La nueva tabla se vería:

	((	()	($	)(	))	)$
S						

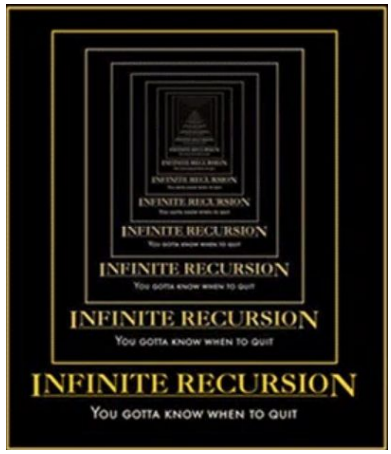
Parsers LL(k)

- El caso general corresponde a un parser de tipo LL(k).
- Se tiene, sorpresa sorpresa, *k* simbolos para decidir la regla a utilizar
- Se tienen que definir las funciones:
    - $FIRST_k()$
    - $FOLLOW_k()$
    - $PREDICT_k()$
- Las tablas van a tener $n^k + n^{k-1}$ columnas, con n terminales en la gramática.
- Por dicha para nosotros, la gran mayoría de lenguajes de programación son LL(1), o se pueden usar otros tipos de parser (como LALR).

Análisis Sintáctico - Descenso Recursivo

C.C. El semestre (ba dum tss)

¿Recuerdan MICRO?



- El primer proyecto del curso fue realizar un compilador completo para el lenguaje MICRO.
- MICRO era LL(1), pero… ¡no hicimos una tabla!
- Lo que hicimos, sin saberlo, fue utilizar una técnica de compilación llamada *Descenso Recursivo*, que es equivalente a LL(1).
- Si se puede hacer por LL(1), se puede hacer por descenso recursivo, y viceversa.
- Ambas técnicas tienen los mismos requisitos.

Descenso Recursivo

- En su forma más básica, se toma la gramática y se convierte en código.
- Siempre hay un símbolo actual de la hilera que se está parseando.
- Cada no terminal tiene un procedimiento asociado.
- Dentro del procedimiento de cada no terminal, se escoge la siguiente regla a utilizar dependiendo del símbolo actual.
- El lado derecho de cada regla se convierte en una serie de llamadas a los procedimientos de los no terminales o un *match* del símbolo actual con el terminal que se espera.

Ejemplos de Descenso Recursivo

Ejemplo 1

E → *i* E'
E' → + *i* E' | ε

```
E(){
    match('i');
    E'();
}

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

match(char t) {
    if(t == c)
        c = gettoken();
    else
        printf("ERROR");
}

main(){
    c = gettoken();
    E();
}
```

Ejemplo 2

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

```
S(){
    match('<statement>');
}

SQ(){
    S();
    SQ'();
}
```

```
SQ'(){
    if (c == ';') {
        match(';');
        SQ();
    } else
        return
}

match(char t) {
    if(t == c)
        c = gettoken();
    else
        printf("ERROR");
}

main(){
    c = gettoken();
    SQ();
}
```

Ejemplo 3

S → IFS | <other>
IFS → if (EX) S EL
EL → else S | ε
EX → 0 | 1

```
S(){
    /* Por fuerza hay que fijarnos, es aqui donde el descenso
    recursivo nos empieza a pasar factura*/
    if (c == 'if')
        IFS();
    else
        match('<other>');
}

IFS() {
    match('if');
    match('(');
    EX();
    match(')');
    S();
    EL();
}
```

```
EL(){
    if(){
        match('else');
        S();
    } else
        return;
}

EX() {
    if (c == '0')
        match('0');
    else
        match('1');
}

match(char t) {
    if(t == c)
        c = gettoken();
    else
        printf("ERROR");
}

main(){
    c = gettoken();
    S();
}
```

Análisis Sintáctico - Mensajes de Error

Errores y LL(1)

- El algoritmo LL(1) es capaz de detectar hileras inválidas, en la tabla de parsing corresponde a las casillas vacías.
- Estrategia simple:
  - “Error de sintaxis aquí”
  - Termina el parseo, no se revisa el resto
- Tratemos de buscar un mejor enfoque:
  - Mensajes de error: hay que dar el mejor diagnóstico posible de la causa del error.
  - Recuperación de error: hay que volver a sincronizar el autómata para poder reportar otros errores en la hilera.

Mensajes de Error

- Es un arte.
- Distintos compiladores dan mensajes diferentes para el mismo error.
- Múltiples posibles causas.
- Particular para cada lenguaje, y para la situación.
- A veces son de manera incremental:
  - Darle un número a cada error.
  - Identificar donde ocurre el error y mejorar la redacción del mensaje.
- Preferiblemente, que sus mensajes no sean muy *malparidos* como:
  - ¡Devuélvase a la U Letrina!
  - ¡Inutil!
  - ¡Izcar!

En este momento, le hicieron un chiste muy muy cruel al pobre Izcar, que ya de por si lo tenían de pato desde temprano en la clase, entonces Torres se compadeció y nos dejó salir.

Recordatorios rápidos

- ¡Proyecto! (31/5/17)
- ¡Exámen! (2/6/17)
- ¡Final! (9/6/17)
- ¡Quiz! (24/5/17)
- ¡Tarea de MICRO! (24/5/17)