APUNTES DEL 19 DE MAYO DEL 2017

Compiladores e Intérpretes

Instituto Tecnológico de Costa Rica.
Ingeniería en Computación.
I Semestre.

Ximena Bolaños Fonseca.
2015073844

Avisos Importantes

Cronograma

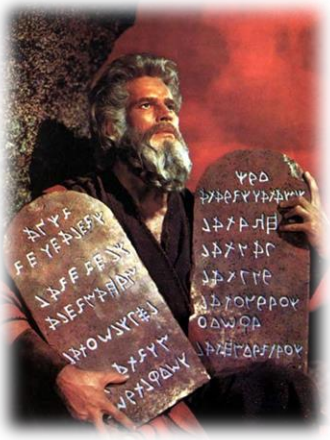
***Solo quedan 3 Clases*!!!!**

Lunes	Martes	Miércoles	Jueves	Viernes
22	23	24 Tarea Corta Tabla de Parsing Para Micro	25	26
29	30	31 Proyecto 3	1	2 II Parcial
5	6	7	8	9 Examen Final

***Si no vinieron a clase y no han sido apuntadores, NO se les Olvide!!!**

Análisis Sintáctico

Construcción de la Tabla de Parsing



- Tabla M
  - Una fila por cada **No Terminal**
  - Una columna por cada **terminal** y **$**
- Se calculan los **FIRST ()** y los **FOLLOW ()**
- Se calcula el **PREDICT** de cada regla
- Para toda regla **A→α**:
  - Para cada elemento de a en **PREDICT(A→α)** agregue la regla **A→ α** en **M [A][a].**

Ejemplo 1

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

	(	)	$
S	1	2	2

Ejemplo 2

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

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

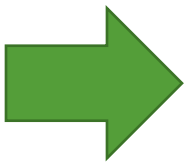
Ejemplo 3

SQ→S; SQ

SQ→S

S→<statement>

Factorizar



SQ→S SQ'

SQ'→; SQ|ε

S→<statement>

FIRST ()

No terminal	Pasada 0	Pasada 1	Pasada 2	Pasada 3
SQ	{}		{<statement>}	
S	{}	{<statement>}		
SQ'	{}	{;, ε}		

FOLLOW ()

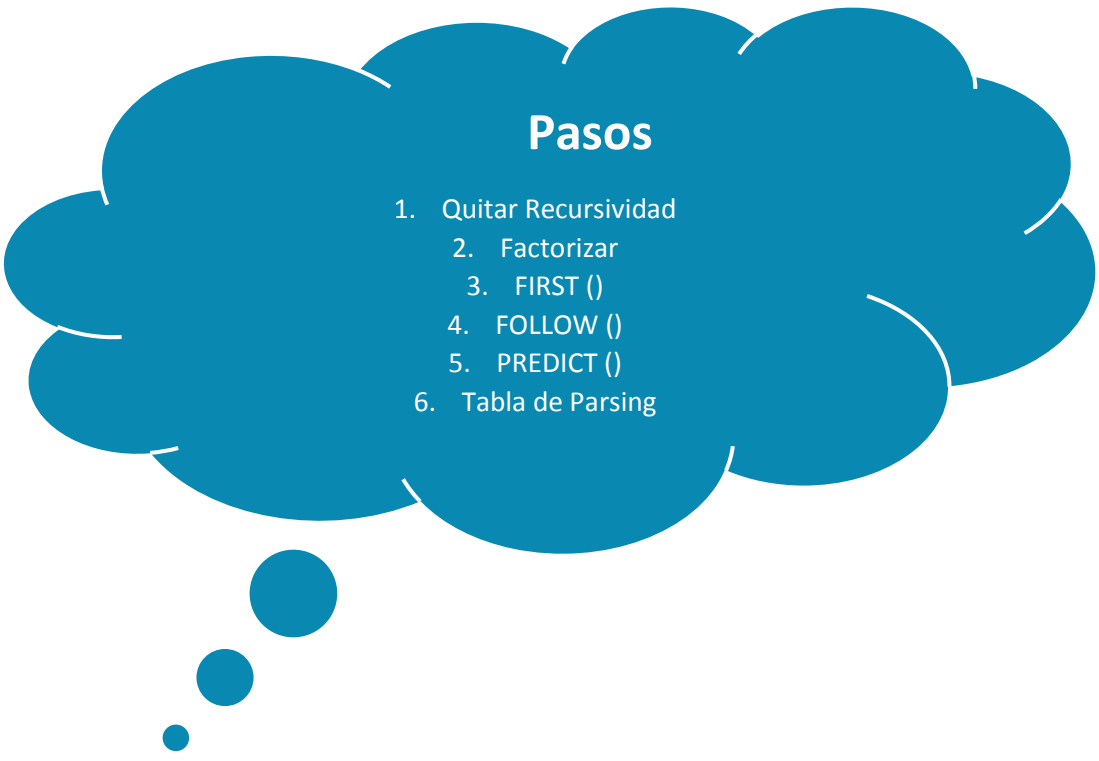
No terminal	Pasada 0	Pasada 1	Pasada 2
SQ	{$}	{$}	
S	{}	{$}	
SQ'	{}	{;, $}	

PREDICT ()

Regla	FIRST(α)	Predict ()
1. SQ→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(if-then-else)

S->IFS|<other>

IFS->if(EX)S EL

EL->else S| ε

EX->0|1

FIRST ()

No terminal	Pasada 0	Pasada 1	Pasada 2	Pasada 3
S	{}	{other}	{if, <other>}	
IFS	{}	{if}	{if}	
EL	{}	{else, ε}	{else, ε}	
EX	{}	{0,1}	{0,1}	

FOLLOW ()

Recordar: No es sacar el FOLLOW () de la regla, si no de los No terminales de la Regla en la que se está.

No terminal	Pasada 0	Pasada 1	Pasada 2	Pasada 3
S	{$}	{$, else}	{$, else}	
IFS	{}	{$}	{$, else}	
EL	{}	{$}	{$, else}	
EX	{}	{)}	{)}	

PREDICT ()

Regla	FIRST(α)	Predict ()
1. SQ→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}

1. Conflicto de Reglas
2. Ambigüedad
3. ¿Qué hacemos?

Tabla de Parsing

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

Ambigüedad del if-then-else

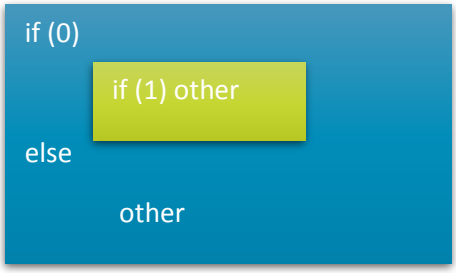
Esta gramática no puede ser procesada por este tipo de parser.

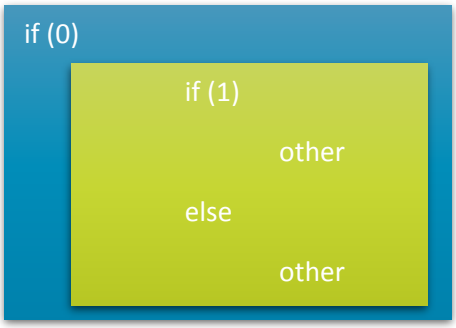
Tiene ambigüedad que significa que al menos existen dos árboles de derivación para esta gramática.

- Considere el código

  if (0) if (1) other else other

- Con la gramática vista, hay 2 posibilidades y ambas son válidas.





- ¿Cuál es la correcta? Ambas son válidas pero la de la derecha sería más natural.

Para arreglarlo hay que manosear el Parser para arreglarlo y haga lo que nosotros queremos.

****El lenguaje Izcar, el if anidado funciona de esta manera.***

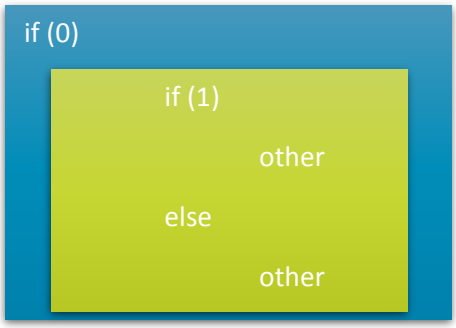
¿Por qué? Porque Izcar lo dijo.

Es importante denotar que no nos volamos la regla solo manipulamos la tabla para que se use de la manera que queremos.

No cambiar el lenguaje, por eso {} no es válido porque cambiaría el lenguaje.

- Problema del “dangling else” (el else que queda guindando).

- Se resuelve forzando que el else se asocie con el if abierto más cercano.



Eliminando Ambigüedad del if-else-then

- Analizar el lenguaje
- Analizar la gramática

PREDICT ()

Regla	FIRST(α)	Predict ()
1. SQ→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}

Tabla de Parsing

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

Resumen

- Se toma CFG
- Elimina la recursividad por la izquierda
- Factorizar por la izquierda
- Calcular FIRST () para todos los No terminales
- Calcular FOLLOW () PARA TODOS LOS No terminales
- Calcular PREDICT () para todas las reglas.
- Construir tabla de parsing.
- No todas las gramáticas permiten este proceso.

Parsing LL (1)

¡Ya lo vimos!

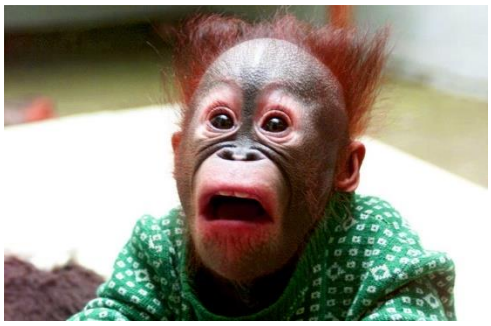
- Tipo de Parsing Top Down
- Es predictivo. No hay backtracking.
- Se llaman LL (1) porque:
    - L→ “Left to Right” el texto se procesa de izquierda a derecha.
    - L→ “Left most” produce una derivación Left most.
    - 1→ se usa un lookahead de 1 token.
- Estudiados en 1968 por Lewis y Steams.

Gramática LL (1)

- Definición
    - Una CFG es LL (1) si para cualquier par de reglas diferentes A→α y A→β se cumple que PREDICT (A→α) ∩ PREDICT (A→ β) = {}.

Es decir, hay una tabla de parsing con solo una regla por casilla. Si no son LL (1) la macheteamos.

- 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
- No todas son LL (1)



No hay que ir muy lejos para buscar uno más complicado, por ejemplo, el lenguaje natural. PINKER

¿Sabían que somos máquinas químicas?

Bison maneja Lenguajes de Programación actuales con gramáticas LALL1. Hay que ser agradecidos con Bison que resuelve los conflictos shift reduce (se traga los conflictos, hay que esforzarse para que no se los trague).

Propiedades de Parsers LL (1)

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

1. Si una hilera no se rechaza, se garantiza que obtendremos 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.

¿Backus entonces uso esta estructura? NO (1953). Backus hizo magia, un código cochino que funcionaba. Ya que estos Parsers se crearon aproximada por el año 1968.

Agradecer cada vez que usen el compilador.

Parsers LL (2)

- Un parser LL (1) puede predecir la regla que debe usarse con solo un símbolo de lookahead
- Hay gramáticas que no son LL (1), aunque no sean recursivas por la izquierda y están factorizadas.
- Puede ser que viendo 2 símbolos por adelantado se pueda predecir la regla que deba usarse.
- Parsing LL (2)
- Se definen las funciones
  - $FIRST_2()$
  - $FOLLOW_2()$
  - $PREDICT_2()$
- Cálculo son mucho más caros.
- La tabla LL (2) tiene (n^2+n) columnas.

Tabla de Parsing LL (2)

1. S→(S)S
2. S→ **ε**

****Ahora hay Parejitas**

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

Parsers LL(k)

- Caso general: parsers LL(k)
- Un lookahead de k símbolos para decidir que regla usar.
- Hay:
  - $FIRST_K()$
  - $FOLLOW_K()$
  - $PREDICT_K()$
- 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 pueden usar otro tipo de parsers.

Descenso Recursivo

¿A que les suena? A mi semestre

- Básicamente se toma la gramática y esta se convierte en código.
- Siempre hay un símbolo actual de la hilera que se está parseando.
- Dentro del procedimiento de cada No terminal se escoge cual regla de la gramática aplicar, dependiendo del símbolo actual.
- El lado derecho de cada regla de la gramática se convierte en llamadas a los procedimientos de cada No terminal o un match del símbolo actual con el terminal esperado.

¿Recuerdan Micro?

- Para el primer proyecto de este curso se desarrolló un compilador completo para el lenguaje MICRO.
- La gramática de MICRO era LL (1) pero no se usó ninguna tabla.
- Se usó una técnica de compilación conocida como “Descenso Recursivo”.
- Es equivalente a LL (1)
    - Si una Gramática es LL (1) se puede hacer por descenso recursivo y viceversa.
- Ambas técnicas tienen los mismos requisitos. **Solo que esta técnica es más cool. Pero en complejidad y eficiencia es lo mismo.**

Si les da tiempo si, lo hacen en Descenso Recursivo la tarea y hacen de nuevo la primera progra. ☺

No les tengan miedo a las ideas Locas porque el Titanic fue construido por profesionales y el Arca de Noé por aficionados. ¿Cuál se hundió?

Ejemplo de Descenso Recursivo

- Noten que por cada No Terminal hay un Procedimiento.
- Dentro de cada uno ustedes escogen que reglas se van a llamar.

```
E→ i E'
E'→ i E'| ε
```

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

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

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

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

Ejemplo 2

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

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

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

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

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

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

Ejemplo 3

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

Aquí es donde ya le sale caro el descenso recursivo se tiene que saber el FIRST () de IFS.

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

```
match (char t)
{
        If(t==c)
                C=gettoken ();
        Else
                Printf("ERROR");
}
```

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

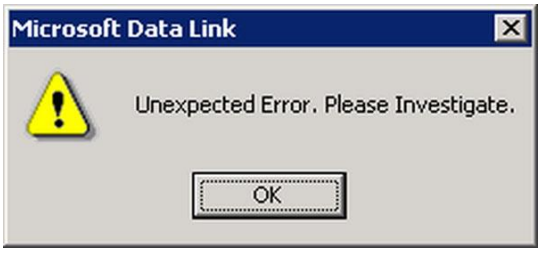
```
EX ()
{
        if(c=='0')
                match ('0');
        else
                match ('1');
}
```

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

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

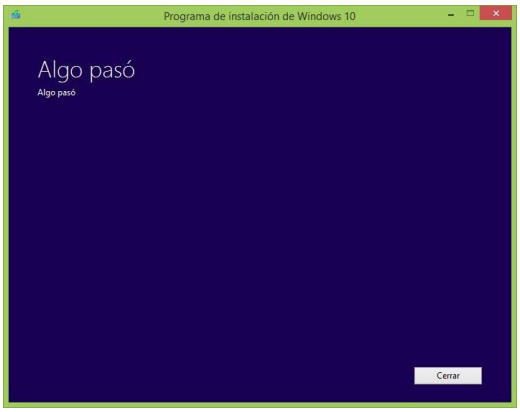
Mensajes de Error

Errores y LL (1)



- El algoritmo detecta hileras inválidas menos la 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 hilera.

Mensajes de Error



- Es casi un arte.
- Distintos Compiladores dan mensajes diferentes para el mismo error.
- Hay muchas causas posibles.
- Es particular para cada lenguaje y situación. Como, por ejemplo, el mismo código corriendo en Ubuntu y Windows.
- A veces se hacen de manera incremental:
  - Dar solo el número de error (ERROR 17).
  - Ver ejemplos del error e ir mejorando la redacción.

Usted no quiere que le salgan Mensajes de Error como:

¡Inútil! ¡Vuelva a la Latina! ! ¡Es un Izcar!

Cierre Doloroso