



**Instituto Tecnológico de Costa Rica**

**Escuela de Computación**

**IC5701 Compiladores e Interpretes**

**Apuntes de clase**

**Fecha: 24 de mayo del 2017**

**Realizado por:**

**Adrián Jesús Álvarez Calderón**

**2014071059**

**Profesor:**

**Francisco José Torres Rojas**

**I Semestre 2017**

## Tabla de contenido

Quiz 11 .....	3
Análisis Sintáctico:.....	3
Errores y LL (1): .....	3
Mensajes de Error:.....	3
Mensajes de Error – Ejemplo: .....	4
Recuperación de error .....	8
Acciones de Recuperación de Error .....	8
Ejemplo de acciones de recuperación .....	9
Ejemplo 1 .....	10
Ejemplo 2 .....	11

## Quiz 11

1. Considere las siguientes 3 gramáticas. Haga todo el proceso necesario para construir la tabla de parsing correspondiente a cada una.

a)  $S \rightarrow aSdd \mid A$

$A \rightarrow bAc \mid bc$

b)  $S \rightarrow 1 \mid 1S1 \mid 0S0$

## Análisis Sintáctico:

... continuando con el manejo de errores.

### Errores y LL (1):

El algoritmo LL (1) tiene el objetivo de detectar hileras inválidas, que corresponden a una entrada vacía en la tabla de parsing LL (1).

Existe una estrategia simple:

- Decir “Error de sintaxis aquí \_\_\_\_” como mínimo.
- No se llega a revisar el resto de la hilera dado un error de sintaxis.

Busquemos un mejor enfoque:

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

### Mensajes de Error:

Crear mensajes de error es casi un arte.

Es común que distintos compiladores dan mensajes diferentes para el mismo error.



Estos errores se pueden ocasionar por múltiples posibles causas.

Son particulares para cada lenguaje, y para una situación en específico.

A veces se dan de manera incremental:

- Darle un número a cada error.
- Se identifica cómo y dónde ocurre el error con el objetivo de ir mejorando la redacción de los mensajes de error.

### Mensajes de Error – Ejemplo:

Considere la siguiente gramática:

1.  $S \rightarrow a$
2.  $S \rightarrow ( S R$
3.  $R \rightarrow , S R$
4.  $R \rightarrow )$

Esta gramática genera hileras similares a programar en LISP, no obstante, se le agregan comas para que sea más interesante el ejercicio.

Esta gramática genera “S-expressions”, es decir hileras de la forma:

- $a$
- $(a)$
- $(a, a, a)$
- $(a, (a, a) )$
- $( (a, (a, a), (a, a) ), a)$

Cuando se quiere llegar a mostrar buenos errores de sintaxis es imperativo entender qué hileras genera la gramática en cuestión. Esta gramática está asociada a la siguiente tabla de parsing:

	a	,	(	)	\$
S	1		2		
R		3		4	
\$					😊

En la cual las celdas vacías representan errores de sintaxis.

En general una opción genérica para mostrar los errores podría ser la siguiente:

“Aparece \_\_\_\_\_ cuando se esperaba \_\_\_\_\_”

Ese tipo de mensaje es particularmente útil para el caso de match, en donde se indica que símbolo es esperado y el símbolo que se estaba recorriendo en el momento. No obstante, no funciona del todo bien cuando lo que está en el tope de la pila es un *No-Terminal*, dado que este concepto de “No-Terminal” es algo muy propio de la gramática interna de lenguaje, puede que no sea de mucha utilidad para el desarrollador que entiende más los conceptos Terminales que los No-Terminales.

Aparece una alternativa para diseñar de una mejor manera los mensajes de error. Esta consiste primeramente en identificar cada error, para ello se asigna una letra del abecedario a cada uno, nótese la tabla siguiente.

	a	,	(	)	\$
S	1	a	2	b	c
R	d	3	e	4	f
\$	g	h	i	j	😊

Ahora, analicemos un poco los errores en particular:

- **Errores a y b:** Se esperaba un “S-expression” y estos inicios que corresponde a los símbolos “,” y “)” no son válidos según la gramática, y corresponden a entradas vacías en la tabla de parsing.

Algunos mensajes apropiados podrían ser:

- *Aparece ‘,’ cuando se esperaba S-expression.*
- *Aparece ‘)’ cuando se esperaba S-expression.*
- **Error c:** Es un error diferente al anterior, primero debemos considerar que hileras generan este error, por ejemplo
  - ( \$
  - ( a, \$
  - \$

Por lo que un mensaje apropiado podría llegar a ser:

*“S-expression incompleta”, “Programa incompleto”*

- **Error d, e, f:** Son errores asociados al No-Terminal R, no obstante R no tiene un significado evidente para los desarrolladores y menos aún que este No-terminal ni siquiera era original de la gramática, sino que proviene de algunas modificaciones necesarias como factorización o eliminación de recursividad por la izquierda en gramáticas. Entonces un buen mensaje de error podría incluir tokens esperados.

*“‘a’ aparece cuando se esperaba ‘,’ o ‘)’”*

*“‘(’ aparece cuando se esperaba ‘,’ o ‘)’”*

*“Fin de hilera aparece cuando se esperaba ‘,’ o ‘)’”*

Nótese que las hileras que típicamente causan los errores d y e son de la forma:

- ( a **a** ) \$
- ( (a) **a** ) \$
- ( a ( a ) ) \$
- ( (a) ( a ) ) \$

Dado este comportamiento se puede deducir que un mejor mensaje de error podría ser: *"Falta una ',' "*.

- **Error g, h, i, y j:** Estos errores suceden cuando el autómata considera que ya la hilera debe terminar y aún quedan símbolos. Nótese las siguientes hileras que generan errores:
  - ( a ) *a* \$
  - ( ( a ), a ) , a ) \$
  - ( a ) ( a ) ) \$

Dado este contexto, un mensaje de error apropiado por mostrar para todos estos casos podría ser:

*"\_\_\_\_\_ aparece después de fin de S-expression"*

Llegado este punto consideremos crear una tabla de mensajes de error para cada error en específico, como punto de referencia para cuando suceda un error poder consultarlo aquí:

Error	Mensaje
<b>a</b>	Aparece ',' cuando se esperaba S-expression.
<b>b</b>	Aparece ')' cuando se esperaba S-expression.
<b>c</b>	S-expression incompleta
<b>d</b>	Falta ','
<b>e</b>	Falta ','
<b>f</b>	Falta ')'
<b>g</b>	'a' aparece después de fin de S-expression
<b>h</b>	',' aparece después de fin de S-expression
<b>i</b>	('' aparece después de fin de S-expression
<b>j</b>	')' aparece después de fin de S-expression

## Recuperación de error

Se debe de reestablecer la pila y avanzar en la entrada para poder seguir analizando el código y encontrar más errores, como se había mencionado hace algunas clases de debe de “saltar a un punto seguro”, el cual preferiblemente no debería de incluir el scope del error encontrado. Esta recuperación nunca va a ser 100% segura y precisa, sino se trabaja bajo un conjunto de heurísticas.

En un caso extremo se llega a dar sólo un mensaje de error, lo cual baja considerablemente la productividad, dado que si es un software lo suficientemente complejo y pesado para tomar varios minutos en compilación para que al final muestre sólo un error de una cantidad  $n$  de errores se vuelve incómodo el desarrollo lo cual incide directamente en que la productividad disminuya.

Razones por las cuales se genera un conjunto de riesgos, entre los principales se destacan la cascada de errores (particularmente cuando no se da un buen “salto a un punto seguro”), puede llegar a ser posible dar mensajes de error equivocados, y por lo general el desarrollador se concentra en los primeros 2 o 3 errores.

## Acciones de Recuperación de Error

Estas acciones de recuperación están directamente asociadas a cada situación en particular, pero también son específicas para cada gramática y lenguaje. En general se usa la tabla de errores (anteriormente mencionada, pero con un agregado). Es importante definir el siguiente conjunto de operaciones por realizar para la recuperación de errores:

- **POP:** Saca el No-Terminal del TOP de la pila.
- **PUSH:** Inserta los elementos indicados en el TOP de la pila.
- **REPLACE:** Reemplaza el TOP de la pila por los elementos indicados.
- **ADVANCE:** Avanza en la entrada.
- **RETAIN:** Permanece en el mismo punto de entrada.
- **EXIT:** Termina el parsing.



## Ejemplo de acciones de recuperación

La tabla de acción viene por magia:

Error	Mensaje	Acción
<b>a</b>	Aparece ',' cuando se esperaba S-expression.	POP-RETAIN
<b>b</b>	Aparece ')' cuando se esperaba S-expression.	POP-RETAIN
<b>c</b>	S-expression incompleta	EXIT
<b>d</b>	Falta ','	REPLACE(R S) - RETAIN
<b>e</b>	Falta ','	REPLACE(R S) - RETAIN
<b>f</b>	Falta ')'	EXIT
<b>g</b>	'a' aparece después de fin de S-expression	PUSH(R S) - RETAIN
<b>h</b>	',' aparece después de fin de S-expression	PUSH(R) - RETAIN
<b>i</b>	('' aparece después de fin de S-expression	PUSH(R S) - RETAIN
<b>j</b>	')' aparece después de fin de S-expression	ADVANCE

Ocupamos además la tabla de parsing:

	<b>a</b>	<b>,</b>	<b>(</b>	<b>)</b>	<b>\$</b>
<b>S</b>	<b>1</b>	<b>a</b>	<b>2</b>	<b>b</b>	<b>c</b>
<b>R</b>	<b>d</b>	<b>3</b>	<b>e</b>	<b>4</b>	<b>f</b>
<b>\$</b>	<b>g</b>	<b>h</b>	<b>i</b>	<b>j</b>	😊

Y claro, la gramática.

1.  $S \rightarrow a$
2.  $S \rightarrow ( S R$
3.  $R \rightarrow , S R$
4.  $R \rightarrow )$

### Ejemplo 1

Dada la gramática y tablas anteriores, debe parsear la siguiente hilera:

( a , , a \$

Pila	Hilera	Log
\$ <b>S</b>	( a , , a \$	Exp R#2
\$ R S (	( a , , a \$	Match
\$ R <b>S</b>	<b>a</b> , , a \$	Exp R#1
\$ R <b>a</b>	<b>a</b> , , a \$	Match
\$ <b>R</b>	, , a \$	Exp R#3
\$ R S ,	, , a \$	Match
\$ R <b>S</b>	, a \$	Error
Aparece ',' cuando se esperaba S-expression Acción: POP - RETAIN		
\$ <b>R</b>	, a \$	Exp R#3
\$ R S ,	, a \$	Match
\$ R <b>S</b>	<b>a</b> \$	Exp R#1
\$ R <b>a</b>	<b>a</b> \$	Match
\$ <b>R</b>	\$	Error
Falta ')' Acción: EXIT		

Después de parsear la hilera:

( a , , a  
          1   ↑   ↑   2  
          |   |

1. Aparece ',' cuando se esperaba S-expression
2. Falta '('

## Ejemplo 2

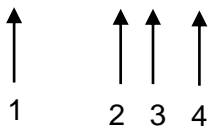
Dada la gramática y tablas anteriores, debe parsear la siguiente hilera:

**((a)a)aa**

Pila	Hilera	Log
\$ <b>S</b>	((a)a)aa\$	Exp R#2
\$ R S (	((a)a)aa\$	Match
\$ R <b>S</b>	(a)a)aa\$	Exp R#2
\$ R R S (	(a)a)aa\$	Match
\$ R R <b>S</b>	a)a)aa\$	Exp R#1
\$ R R <b>a</b>	a)a)aa\$	Match
\$ R <b>R</b>	)a)aa\$	Exp R#4
\$ R )	)a)aa\$	Match
\$ <b>R</b>	a)aa\$	Error
Falta ',' Acción: REPLACE (R S) - RETAIN		
\$ R <b>S</b>	a)aa\$	Exp R#1
\$ R <b>a</b>	a)aa\$	Match
\$ <b>R</b>	)aa\$	Exp R#4
\$ )	)aa\$	Match
\$	aa\$	Error
'a' aparece después de fin de S-expression Acción: PUSH (R S) - RETAIN		
\$ R <b>S</b>	aa\$	Exp R#1
\$ R <b>a</b>	a\$	Match
\$ <b>R</b>	a\$	Error
Falta ',' Acción: REPLACE (R S) - RETAIN		
\$ R <b>S</b>	a\$	Exp R#1
\$ R <b>a</b>	a\$	Match
\$ <b>R</b>	\$	Error
Falta ')' Acción: EXIT		

Después de parsear la hilera:

( ( a ) a ) a a



1 2 3 4

1. Falta ‘,’.
2. ‘a’ aparece después de fin de S-expression.
3. Falta ‘,’.
4. Falta ‘)’.