TEC | Tecnológico de Costa Rica

Compiladores e Intérpretes

Apuntes Compiladores

26-05-2017

Luis Diego Vargas Arroyo

2014036213

Tabla de contenido

Análisis Sintáctico

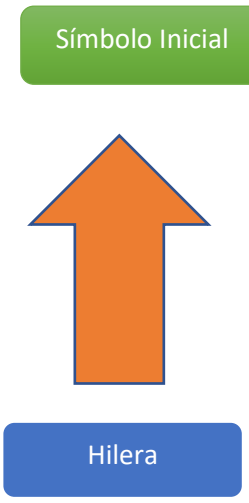
Somos monstruos peludos en parsing LL1

Vamos a ver el parsing botton-up como un par de pinceladas.

Tass.. ….Tasss

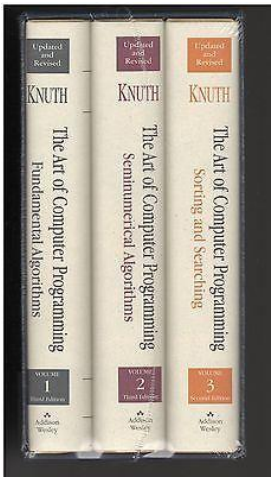
Parsing botton- up

La dirección de parsing es se empieza por la Hilera y se llega al símbolo Inicial



Donald Ervin Knuth

- Matemático y cientifico de la computación USA (1938-)
- “Padre del análisis de Algoritmos”
- Autor de “The art of computer Programing”



Esperamos que el profe los deje olvidados

En una silla un día de estos. 😋

Enorme contribuciones de Knuth en :

- Análisis de algoritmos
- Compiladores
  - Desarrolla la teoría general de Parsing LR (Botton UP)
- Arquitectura de Computadoras
- Creador de Tex (luego LATEX)

- Unas de la personajes más importantes e influyentes en ciencia de la computación.
- Premio turing 1974

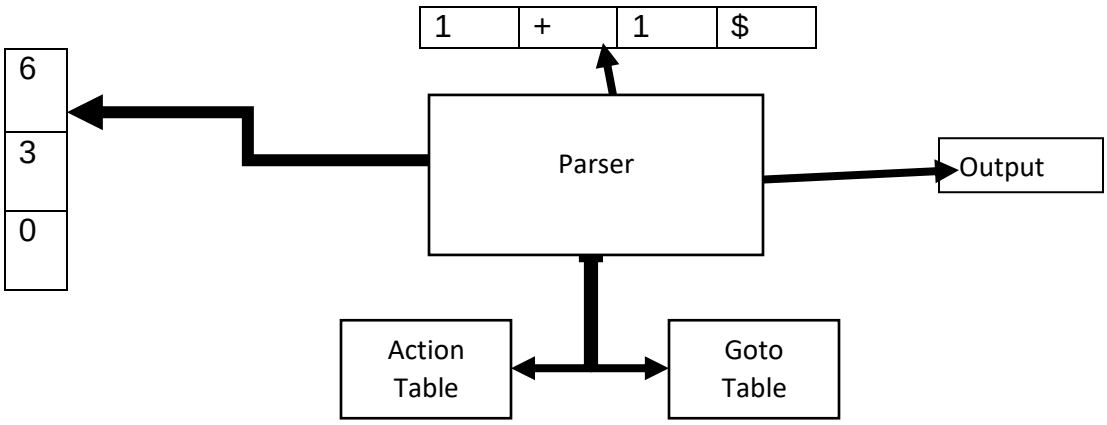
Parsing LR

- Descubierto por Donald Ervin Knuth
- Botton Up parser
- Fue inventado en el año 1965 -> Fue petición del profe que investigaramos el año exacto.

✓ L: el texto se lee de izquierda a derecha sin backtracking

✓ R: si la hilera esta correcta se encuentra la derivación rightmost que la genera

Usualmente LR(K) con k símbolos de lookahead. con K >=0; Si K puede ser 0 predice el símbolo.



- El parser siempre tiene un símbolo actual. Es como un autómata.
- Siempre hay que derivar el terminal que está más a la derecha.

1) $E \rightarrow E+T$
2) $E \rightarrow T$
3) $T \rightarrow T*F$
4) $T \rightarrow F$
5) $F \rightarrow (E)$
6) $F \rightarrow id$

	Action Table						Goto Table		
state	**id**	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

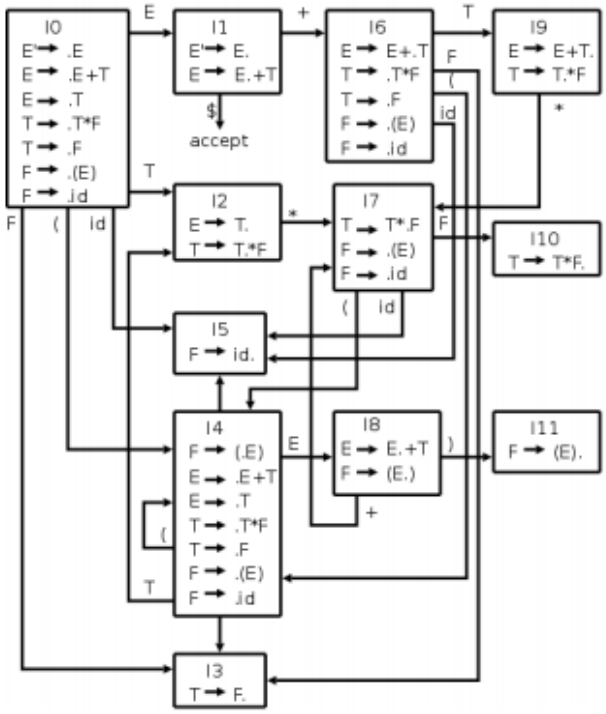
Example LR Parsing

Grammar:

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow \mathbf{id}$

Stack	Input	Action
\$ 0	**id*id+id\$**	shift 5
\$ 0 **id** 5	***id+id\$**	reduce 6 goto 3
\$ 0 *F* 3	***id+id\$**	reduce 4 goto 2
\$ 0 *T* 2	***id+id\$**	shift 7
\$ 0 *T* 2 * 7	**id+id\$**	shift 5
\$ 0 *T* 2 * 7 **id** 5	**+id\$**	reduce 6 goto 10
\$ 0 *T* 2 * 7 *F* 10	**+id\$**	reduce 3 goto 2
\$ 0 *T* 2	**+id\$**	reduce 2 goto 1
\$ 0 *E* 1	**+id\$**	shift 6
\$ 0 *E* 1 + 6	**id\$**	shift 5
\$ 0 *E* 1 + 6 **id** 5	**\$**	reduce 6 goto 3
\$ 0 *E* 1 + 6 *F* 3	**\$**	reduce 4 goto 9
\$ 0 *E* 1 + 6 *T* 9	**\$**	reduce 1 goto 1
\$ 0 *E* 1	**\$**	accept

La construcción seria la siguiente:



Hay que ir siguiendo el punto.

Propiedades de Parsing LR

- Hay algoritmos que automáticamente generan las tablas de parsing LR
- Derivación rightmost encontrada botton UP de manera determinística
- Parser “shift-reduce”
  Problema Tablas enormes(sobre todo para época en que fueron inventados)

Soluciones para el problema de las tablas Gigantescas.

SLR(SImple LR)

- Quita algunos símbolos.
- Hay ciertas gramáticas que no son SLR. Conflicto con el shift-reduce

LALR(Look-ahead LR)

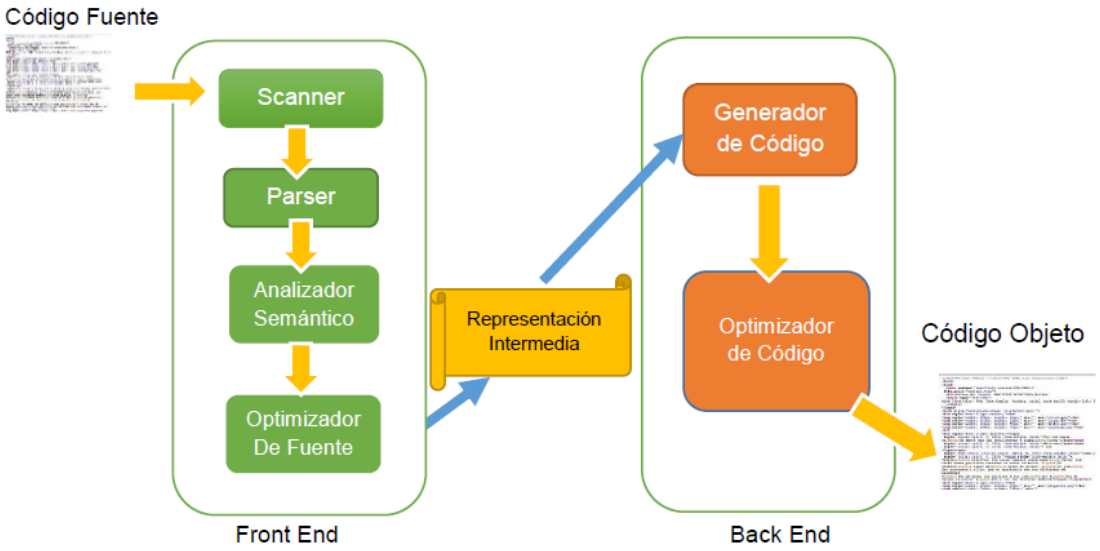
- Alcanza para cualquier lenguaje de programación creado hasta el momento.
- Habría que pensar mucho para que algo no sirva.
- Las tablas no son tan gigantescas

Ambos fueron por:

- Frank DeRemer en 1969

Análisis semántico

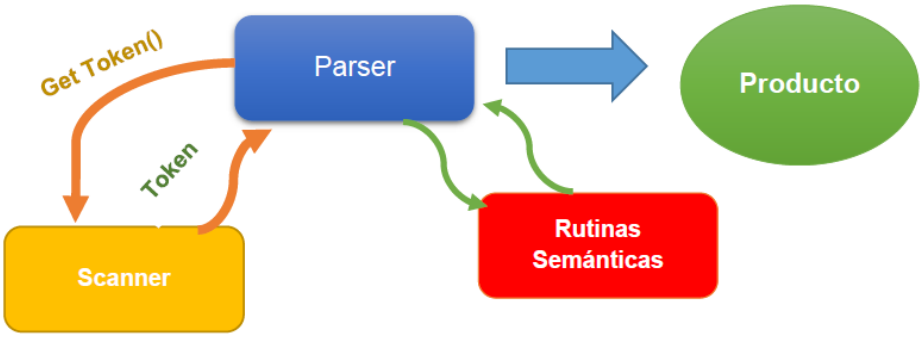
¿Se recuerdan de esta imagen?



Al fin llegamos a análisis semántico y tenemos que ver lo que falta el optimizador de código, la representación intermedia, el generador de código y el optimizador de código en una clase.

Análisis Sintáctico

- Revisar si tiene sentido lo que se pide que se haga.
- Determina su comportamiento en tiempo de corrida
- Declaraciones, revisiones de tipo.
    - Construye la tabla de símbolos.



Acuérdense que la traducción es dirigida por sintaxis, esto quiere decir que el parser es el encargado de llamar las rutinas semánticas.

Errores Semánticos:

- No coinciden los tipos
- Variable no declarada
- Identificador reservado uso indebido.
- Declaración de variables múltiples en un ámbito.
- Acceder a una variable fuera de alcance.
- Parámetro formal y real no coincide.

Actores del Análisis Semántico

- Pila semánticas
- Registros semánticos
- Acciones semánticas

Geografía Gramatical

¿Adónde van las acciones semánticas?

Declaración de variables

Ya no hay slide.

Ejemplo:

En algún lugar del programa dirá;

Int k;

La regla es la siguiente:

DECLARACION -> tipo ID;

Debemos agregar acciones semánticas

DECLARACION -> Tipo # ID;

= Revisar tipo.

Después de que veo al tipo hago algo y sale de la pila semántica y sigue parseando

Optimizador de Fuente

- Introduce optimizaciones
- Cálculo de constantes- Propagación de constantes
- Eliminación de código innecesario
- Se puede hacer sobre árbol anotado
- Es fácil convertir un árbol en estructura lineal
- Similar a un ensamblador de 3 direcciones
- Se genera Código intermedio

Constant folding

Proceso de reconocer y evaluar expresiones constantes en tiempo de compilación para no calcularlas en tiempo de ejecución.

Pueden ser solo constantes:
K = 4 + 3 * 2; ⇨ **k=10;**

O expresiones que den constantes:
K = 0*J; ⇨ **K = 0 ;**
K = J/J; ⇨ **K = 1 ;**

Constant Propagation

Proceso de sustituir los valores de constantes conocidas en tiempo de complicación, se combina con constant folding.

Ejemplo:

```
int x = 14;
 int y = 7 – x / 2;
return y * (28 / x + 2);
```

El compilador lo convierte en:

```
int x = 14;
int y = 7 – 14 / 2;
return y * (28 / 14 + 2);
```

luego en:

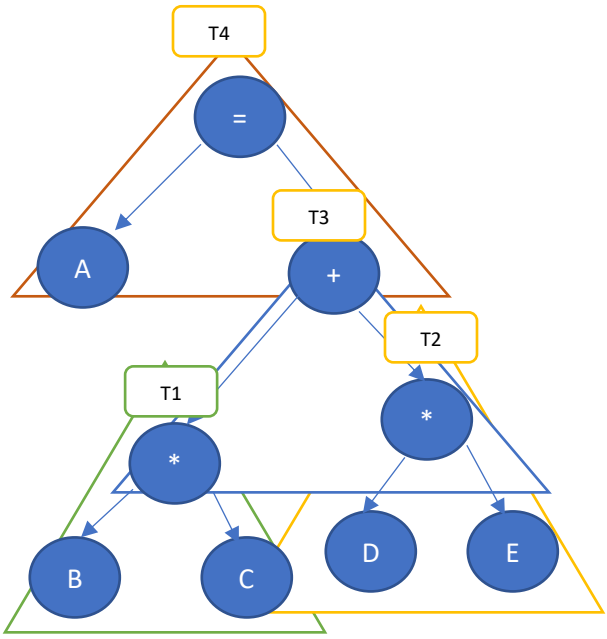
```
int x = 14;
int y = 0;
return 0;
```

Código de 3 direcciones

- Conocido como TAC o 3A
- Cada instrucción tiene a lo más 3 argumentos, pueden ser menos
- Típicamente una asignación y un operador binario
- Muchas variables intermedias temporales
- Comparar contra cero y bifurcar.
- Se puede tener GO TO.

Ejemplo 1

A = B*C+ D*E



T1 = B * C

T2 = D * E

T3 = T1 + T2

T4 = A = T3

Ejemplo 2

if (a < (b + c))

a = a – c;

c = b * c;

T1 = B + C

T2 = A < T1

IFZ T2 L1

T3 = A – C

T4 = A = T3

L1: T5 = B* C

T6 = C = T5

De código Intermedio a Ensamblador

- Número finito de instrucciones de código intermedio
- La eficiencia no se considera todavía.
- Se pueden tener patrones predefinidos en ensamblador para cada una.
- Es relativamente fácil de hacer
- Fácil de cambiar de arquitectura

Ejemplo 1:

Expresión:

A = B * C + D * E

Código 3 direcciones

T1 = B * C

T2 = D * E

T3 = T1 + T2

```
mov ax, c
mul B
mov T1, ax
----------------------
mov ax, E
mul D
mov T2, ax
--------------------------
mov ax, T2
add ax, T1
mov T3, ax
-------------------------
mov ax, T3
mov A, ax
mov T4, ax
```

Ejemplo 2

if (a < (b + c))

a = a – c;

c = b * c;

T1 = B + C

T2 = A < T1

IFZ T2 L1

T3 = A – C

T4 = A = T3

L1: T5 = B* C

T6 = C = T5

```
mov ax, C
add ax, B
mov T1, ax
----------
mov ax, T1
cmp ax, A
setl ax
mov T2, ax
----------
mov ax, T2
cmp $0, ax
jeq L1
----------
mov ax, A
sub ax, C
mov T3, ax
----------
mov ax, T3
mov A, ax
mov T4, ax
----------
L1: mov ax, C
mul B
mov T5, ax
----------
mov ax, T5
mov C, ax
mov T6, ax
----------
```

Investigación del SETL

Setl: set byte if less.

Optimización de código

- Mejora el código generado en el paso previo
- Instrucciones equivalentes, pero más rápidas
- Modos de direccionamiento más apropiadas
- Modos de direccionamiento más apropiados
- Eliminar código redundante
- Optimizar para espacio o para tiempo

Ejemplo 1:

Expresión:

A = B * C + D * E

Código 3 direcciones

T1 = B * C

T2 = D * E

T3 = T1 + T2

```
mov ax, c
mul B
mov T1, ax
----------
mov ax, E
mul D
mov T2, ax
----------
mov ax, T2
add ax, T1
mov T3, ax
----------
mov ax, T3
mov A, ax
mov T4, ax
```

Codigo Optimizado

```
mov ax, c
mul a
mov dx, ax
mov ax, D
mul E
add ax, dx
mov A, ax
```

Ejemplo 2

if (a < (b + c))

a = a – c;

c = b * c;

T1 = B + C

T2 = A < T1

IFZ T2 L1

T3 = A – C

T4 = A = T3

L1: T5 = B* C

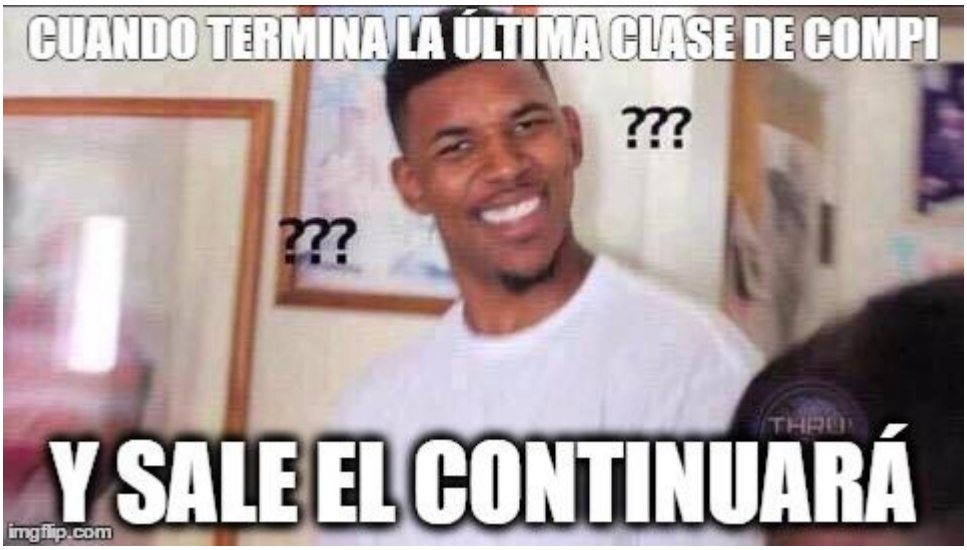
T6 = C = T5

mov ax, C
add ax, B
mov T1, ax
---
mov ax, T1
cmp ax, A
setl ax
mov T2, ax
---
mov ax, T2
cmp $0, ax
jeq L1
---
mov ax, A
sub ax, C
mov T3, ax
---
mov ax, T3
mov A, ax
mov T4, ax

L1: mov ax, C
mul B
mov T5, ax
---
mov ax, T5
mov C, ax
mov T6, ax

Código Optimizado

mov ax, C
mov bx, B
add ax, bx
cmp ax, A
jle L1
mov ax, A
sub ax, C
mov A, ax
L1:
mov ax,c
mul B
mov C, ax



Bueno muchach@s se acabó recuerden:

- Entrega proyecto el 31 de mayo de 2017
- II Parcial el 2 de junio
- Examen Final (Donde entra todo) 9 de Junio