Tecnológico de Costa Rica

San José

Compiladores e Intérpretes

Apuntes del Viernes 26 de Mayo del 2017

César Borge Mora (2015075361)

Profesor Francisco Torres Rojas

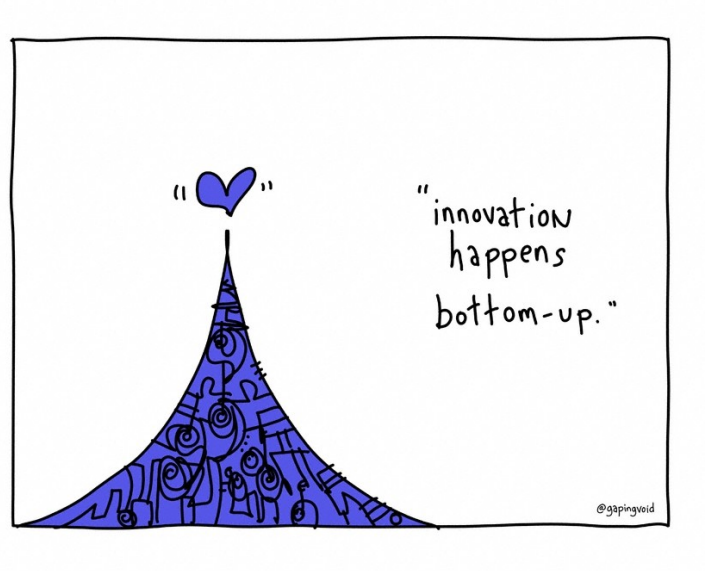
Primer Semestre 2017

Antes de empezar, hay que tener claro que en esta clase metimos el acelerador y vimos prácticamente todas las partes del compilador que nos hacían falta, asi que:

pero

En realidad fueron explicaciones muy resumidas, entonces no hay nada de qué preocuparse, así que empecemos.

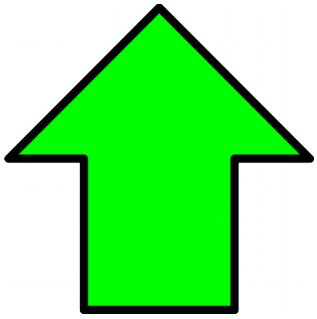
Bottom Up Parsing



aww que cosi

A diferencia del Top Down, se empieza por la hilera y se debe llegar al símbolo inicial:

Símbolo Inicial



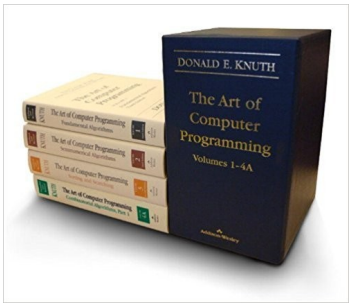
Hilera

Donald Ervin Knuth

Es uno de los más reconocidos expertos en ciencias de la computación por su fructífera investigación dentro del análisis de algoritmos y compiladores.

Knuth es conocido como “El padre del Análisis de Algoritmos”

Es el autor de “The Art of Computer Programming”



Es el creador de Tex (luego LATEX)

En este video Donald Ervin Knuth nos da un consejo a todos los jóvenes: (Trendy != Good)

https://www.youtube.com/watch?v=75Ju0eM5T2c

Por supuesto, en 1974 fue ganador del Premio Turing (La Palangana)

Además, descubrió nuestro siguiente tema:

Parsing LR

Fueron descubiertas por Donald Knuth a mediados de 1960.

Es un tipo de Bottom-Up parser

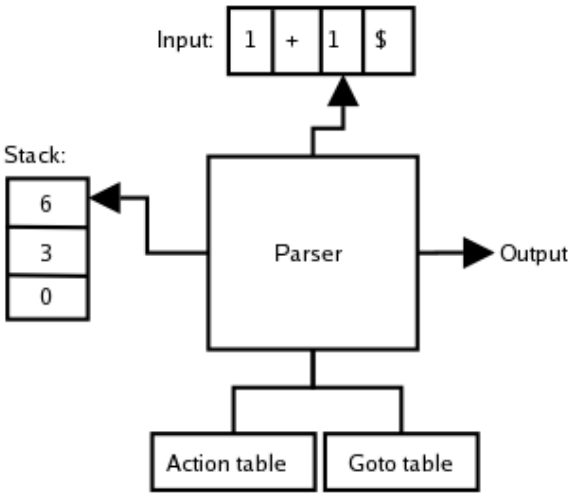
La **L** significa que el texto se lee de **izquierda** a derecha sin utilizar backtracking

La **R** significa que si la hilera es correcta se encuentra una derivación **rightmost** que la genera.

Usualmente se encuentre como LR(k) donde k son la cantidad de símbolos de lookahead, por ejemplo: LR(0), LR(1), LR(2).

Una de las peticiones del profe era la fecha en que Knuth invento el LR:
*The work on LR(k) parsing appeared in a **1965** paper On the translation of languages from left to right.*
Fuente: http://www-history.mcs.st-andrews.ac.uk/Biographies/Knuth.html

Algo así es el Parsing LR:



(SLR) Parsing Tables for Expression Grammar

1) E → E+T
2) E → T
3) T → T*F
4) T → F
5) F → (E)
6) F → id

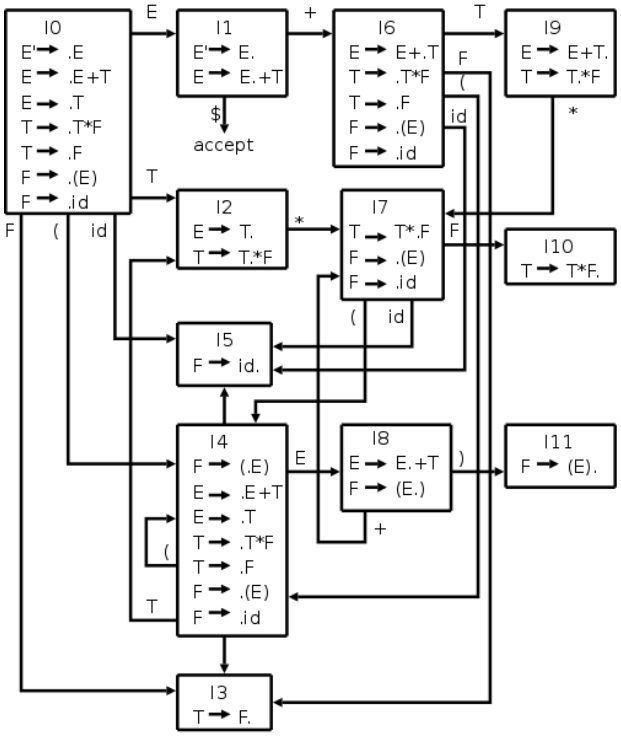
	Action Table						Goto Table		
state	**id**	+	*	(	)	$	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

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. $F \rightarrow$ **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

Construcción:



Propiedades de Parsing LR

Hay algoritmos que automáticamente generan las tablas de parsing LR.

Derivación rightmost encontrada bottom up de manera deterministica.

Es un parser shift-reduce.

Tiene el problema que sus tablas son enormes y es por esto que en la época en que este método fue creado, no recibió mucho apoyo pues los recursos de la epoca no eran suficientes.

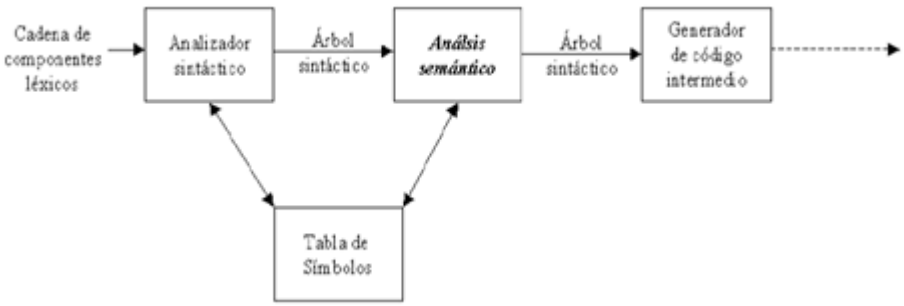
Por esto Frank DeRemer creo **SLR (Simple LR) y LALR (Look-ahead LR).**

Frank DeRemer

Esto dice en la página oficial del hombre:
Frank was raised in a Christian home, but it took a while to "take". He became sold out to Jesus in 1976(A la puña)

Análisis Semántico

Utiliza como entrada el árbol sintáctico detectado por el análisis sintáctico para comprobar restricciones de tipo y otras limitaciones semánticas y preparar la generación de código.



Recordemos que realmente la traducción es dirigida por sintaxis es decir, dirigida por el parser, por ende las rutinas semántica son llamadas por el parser.

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.

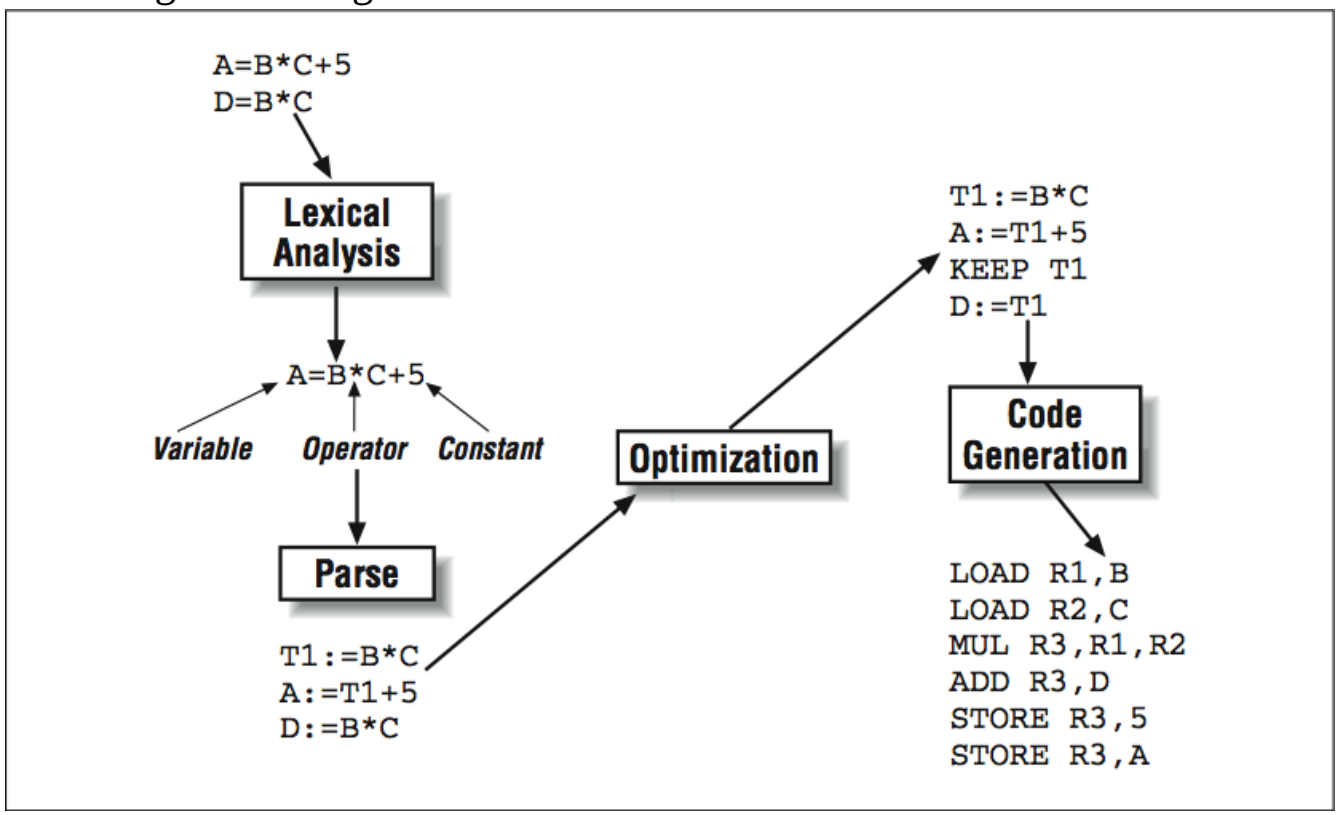
Actores del Análisis Semántico:

- Pila Seáantica
- Registros Semánticos
- Acciones Semánticas

When el profe no deja de explicar cosas nuevas y tu estas asi:

Optimizador de Código Fuente

- Introduce optimizaciones.
- Cálculo de constantes.
- Propagación de constantes.
- Eliminación de código innecesario.
- Se puede hacer sobre arbol anotado.
- Es fácil convertir el arbol en una 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 compilacón para no calcularlas en tiempo de ejecución.

Ejemplos:

Si el compilador encuentra la expresión:

K = 2 + 1;

El compilador convierte esta expresión a:

K = 3;

Expresiones que den constantes como:

K = 0 * J;

El compilador convierte esta expresión a:

K = 0;

Otro ejemplo:

K = J/J;

Lo cambiaría por:

K = 1;

NOTA: El punto flotante NO es nuestro amigo.

Constant Propagation

Proceso de sustituir los valores de constantes conocidad en tiempo de compilación, se combina con constant folding.

Ejemplos:

```
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:

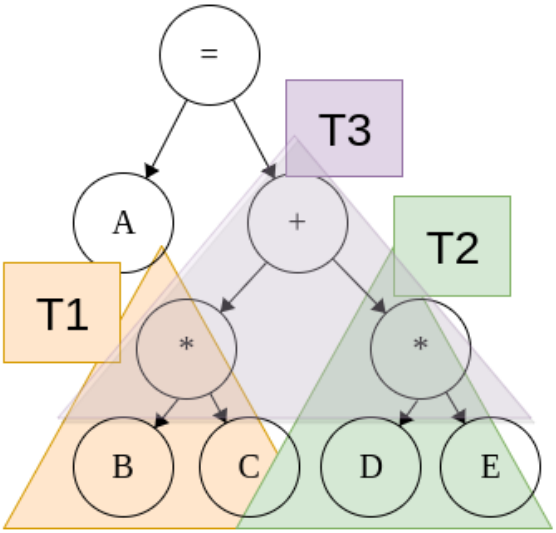
```
int x = 14;
int y = 0;
return 0;
```

Código de 3 Direcciones

- Conocido como TAC o 3AC.
- Cada instrucción tiene a lo mas 3 argumentos – pueden ser menos.
- Típicamente una asignacion y un operador binario.
- Muchas variables intermedias temporales.
- Etiquetas (parecido a ensamblador).
- Comparar contra cero y bifurcar.
- Se vale tener GO TO.

Ejemplo 1:

Expresión: A = B*C + D*E



T1 = B * C

T2 = D * E

T3 = T1 + T2

T4 = A = T3

Ejemplo 2:

Expresión:

```
if (a < (b + c))
        a = a – c;
c = b * c;
```

Código de 3 direcciones:

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

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

Ejemplo 1:

Expresión:

A = B * C + D * E

Código de 3 direcciones:

T1 = B * C

T2 = D * E

T3 = T1 + T2

T4 = A = T3

Código generado:

```
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
```

Optimizador de Código

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

Ejemplo:

Expresión:

```
if (a < (b + c))
        a = a – c;
c = b * c;
```

Código de 3 direcciones:

```
T1 = B + C

T2 = A < T1

IFZ T2 L1

T3 = A – C

T4 = A = T3

L1: T5 = B* C
     T6 = C = T5
```

Código generado:

```
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
```

Y LISTO! FELIZ FIN DE SEMESTRE!

Recordatorios:

- Miércoles 31, entrega de la progra.
- Viernes 2, parcial 2.
- Viernes 9, examen final.