APUNTES 26 DE MAYO

Compiladores e Interpretes

Profesor: Dr. Francisco Torres Rojas

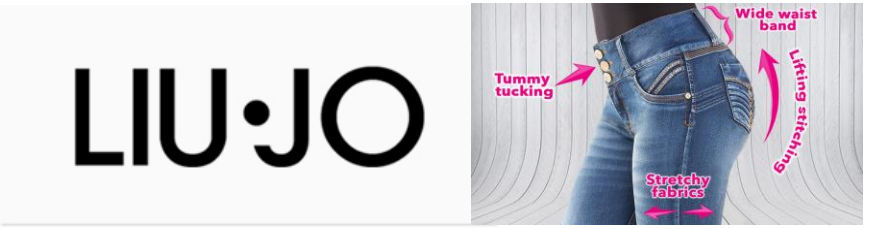
Isaac Campos Mesén

Apuntes 26 de mayo de 2017 – Isaac Campos

Contenido

Apuntes 26 de mayo de 2017 – Isaac Campos

Análisis Sintáctico



La siguiente sección es auspiciada por LIU•JO, de abajo para arriba.

Con la colaboración de Inspector con el tema “Ska Voove Boobye Baby”

<<De arriba para abajo, de abajo para arriba, de abajo para arriba de arriba para abajo>>

Apuntes 26 de mayo de 2017 – Isaac Campos

Parsing Botton UP

Una vez que se define que somos monstruos peludos en parsing LL(1)

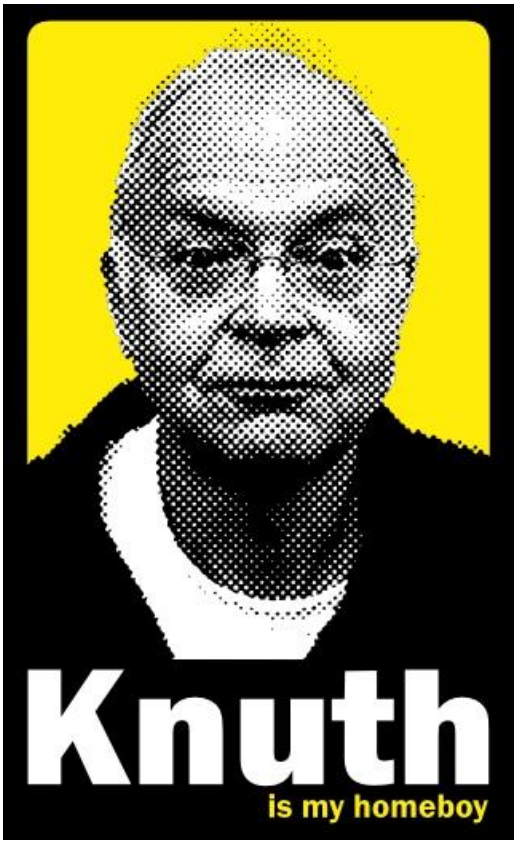
se procede con el tema de Parsing Botton UP

Se llama así por la dirección de Parsing

Se empieza por la hilera y se debe llegar al símbolo inicial

Apuntes 26 de mayo de 2017 – Isaac Campos

Donald Ervin Knuth

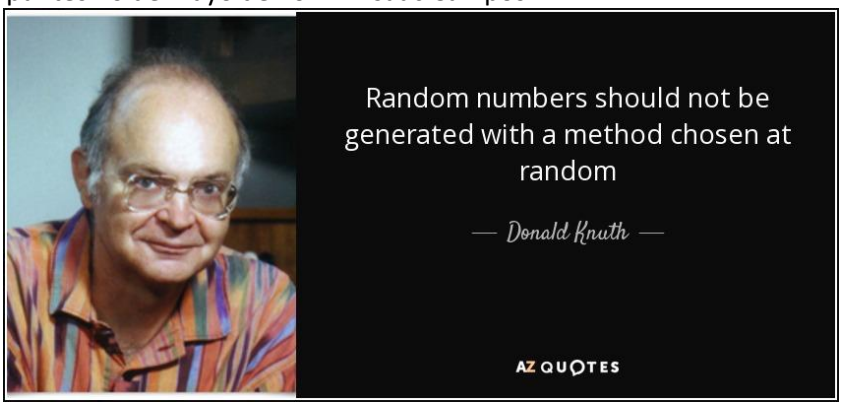


- Matemático y científico de la computación USA 1938 … Actualidad
- "Padre del análisis de algoritmos"
- Autor de "The art of computer programming"

Enormes contribuciones en:

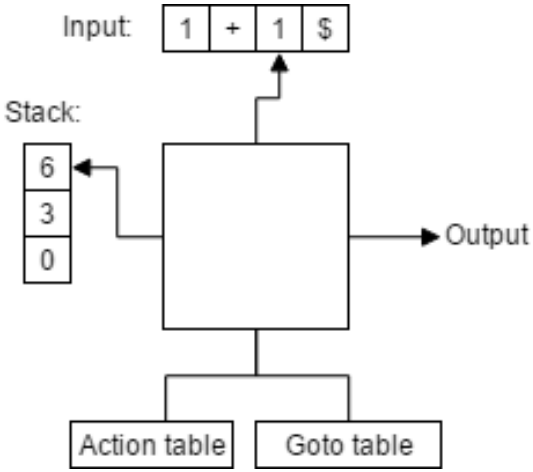
- Analisis de algorimos
- **Compiladores**
  - **Desarrolla la teoría general del parsing LR (bottom up)**
- Arquitectura de computadores
- Creador de TEX (luego LATEX)
- Uno de los personajes más importantes de
- Premio Turing 1974

Apuntes 26 de mayo de 2017 – Isaac Campos



Parsing LR

- Descubiertos por Donald a mediados de los 1960
- Botton-up parser
- L: el texto se lee de **izquierda** a **derecha** sin **backtraking**
- R: si la hilera está correcta se encuentra la derivación **rightmost** que la genera
- Usualmente **LR(k**) con **k** símbolos **lookahead**
    - LR(0), LR(1), LR(2), etc



Apuntes 26 de mayo de 2017 – Isaac Campos

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				

- El parser siempre tiene un símbolo actual.
- Siempre hay que derivar el terminal que está más a la derecha.
- Por ser botton up comienza con 0*
-

Cuando hay conflicto utiliza el shift por heurística

Example LR Parsing

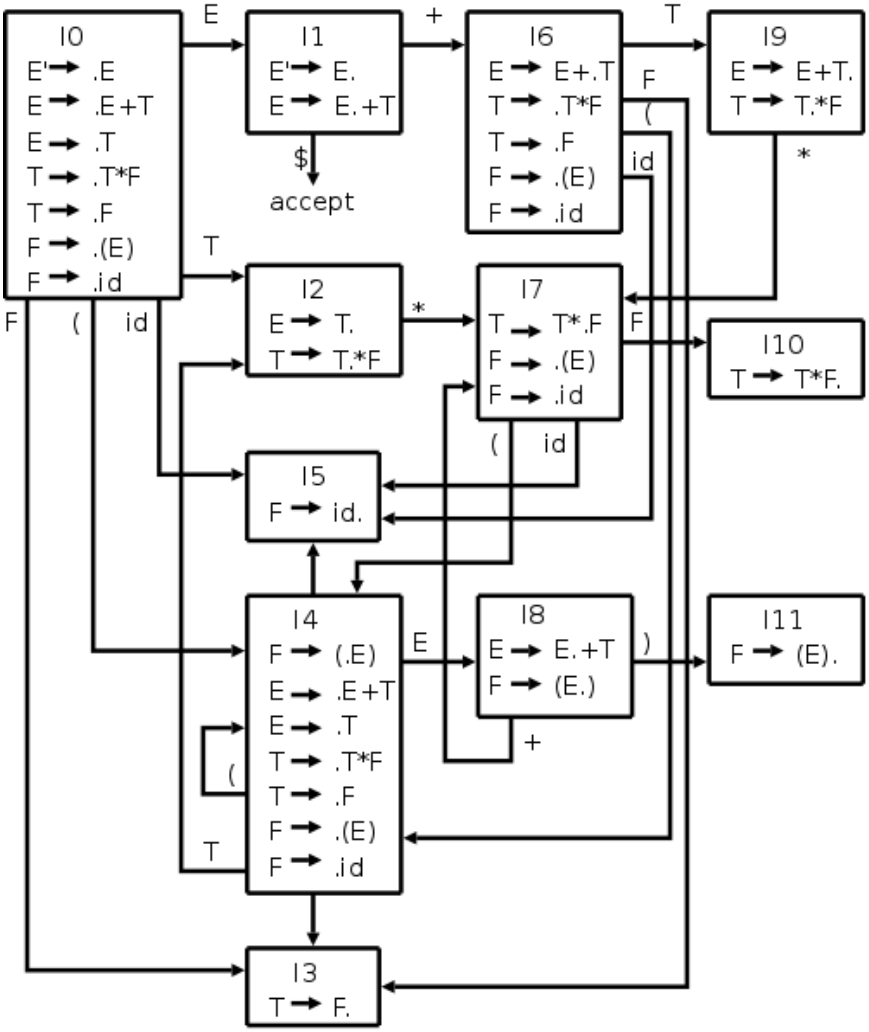
Grammar:

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

Apuntes 26 de mayo de 2017 – Isaac Campos

Construcción de parser LR



Se debe ir siguiendo el punto

Apuntes 26 de mayo de 2017 – Isaac Campos

Propiedades de parsing LR

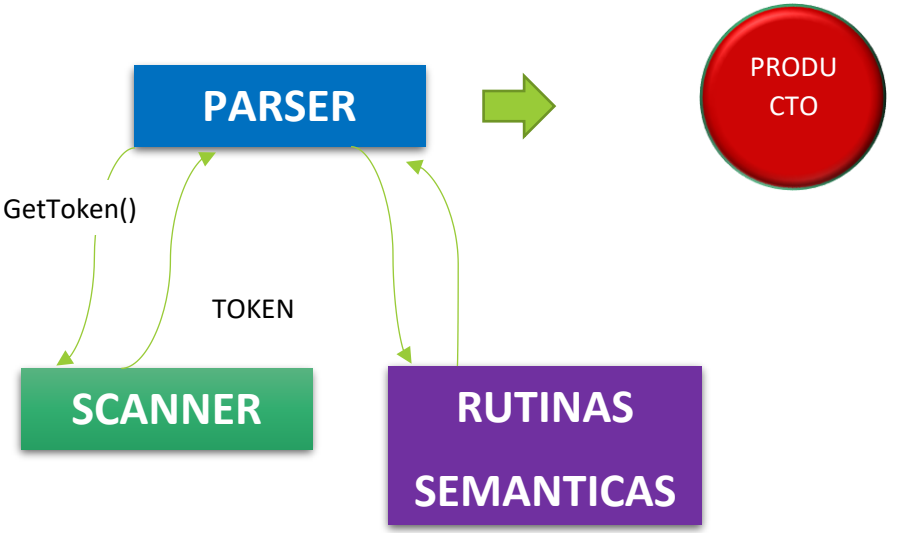
- Hay algoritmos que automáticamente generan las tablas de parsing LR
- Derivación **rightmos** encontrada **botton up** de manera determinísticas
- Parser **“shift-reduce”**
- Problema: tablas enormes (sobre todo para la época que fueron inventados)
- Solución:
    - **SLR (**Simple LR**)**
    - **LALR (**look-ahead LR**) *compatible con cualquier gramática**
    - Creados por Frank DeRemer 1969

Los parsers LR fueron inventados por Donald Knuth in 1965

Apuntes 26 de mayo de 2017 – Isaac Campos

Análisis semántico

- Significado del programa
- Revisa si tiene sentido lo que se pide que se haga
- Determina su comportamiento en tiempo de corrida
- **Semántica estática**
- Declaraciones, revisión de tipos
- Construye **Tabla de Símbolos**
- **Arbol de atributos – árbol decorado**



Actores del Análisis Semántico

- Pila semántica
- Registros Semánticos
- Acciones Semánticas

Geografía gramatical

- ¿Adónde van las acciones semánticas?

Declaración de Variables

- Sale de la pila semántica y sigue parseando

Apuntes 26 de mayo de 2017 – Isaac Campos

Optimizador de fuente

Debido a las técnicas pedológicas de Torres, se justifica como es que todo lo que hemos aprendido en el curso es mentira

- Introduce optimizaciones
- Cálculo de constantes – Propagación de constantes
- Eliminación de código innecesario
- Se puede hacer sobre árbol anotado

Apuntes 26 de mayo de 2017 – Isaac Campos

Constant Folding

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

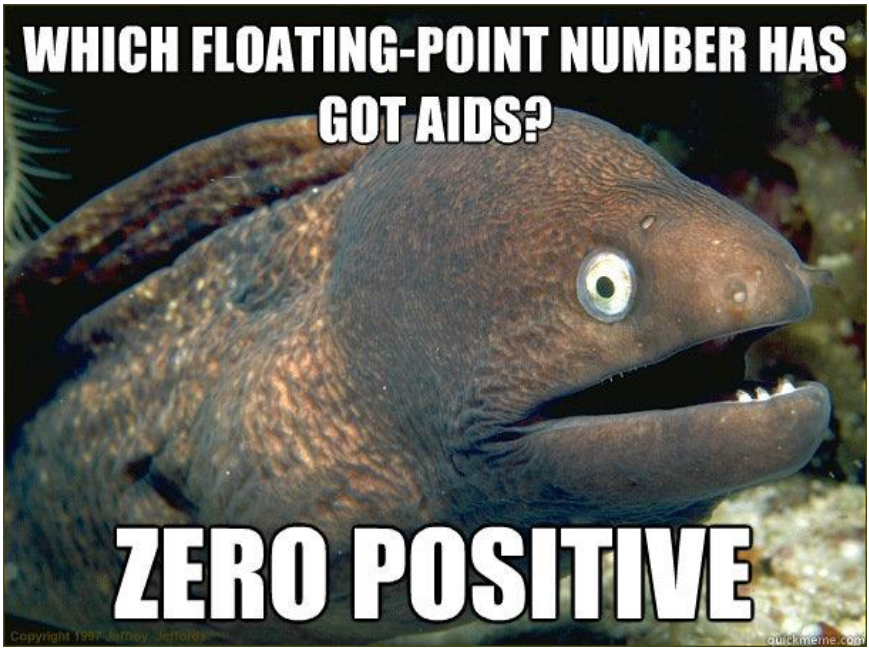
Pueden ser solo constantes:

- K = 4 + 3 * 2; -> k = 20;

O expresiones que den constantes:

- K = 0 * J; -> K = 0
- K = J / J; -> K = 1

*Tener cuidado con punto flotante porque no odia



Apuntes 26 de mayo de 2017 – Isaac Campos

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;
```

Apuntes 26 de mayo de 2017 – Isaac Campos

Código de 3 Direcciones

- Conocido como TAC o 3AC
- Cada intriccuón tiene a lo más **3 argumentos –** puede ser menos
- Típicamente una asignación y n operador binario
- Muchas variables intermedias temporales
- Etiquetas (parecido a ensamblador)
- Comparar contra cero y bifurcar
- Se vale tener GO TO

Ejemplo1:

Expresión: A = B*C + D*E

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

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

Apuntes 26 de mayo de 2017 – Isaac Campos

Generador de código

- Back End del compilador
- Convierte el código intermedio en Código Real
- ¿Ensamblador? ¿Lenguaje máquina?
- Representación de Datos muy importante
    - Cantidad de bytes
    - Restricciones de colocación

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

Ejemplo 1:

Expresión: A = B * C + D * E

Código de 3 direcciones:	Código generado:

T1 = B * C
T2 = D * E
T3 = T1 + T2
T4 = A = T3
	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

Apuntes 26 de mayo de 2017 – Isaac Campos

Optimización de código

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

Ejemplo:

Expresión	Código de 3 direcciones:	Código generado:	Código optimizado:

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
			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