

Autómatas de pila

Pedro O. Pérez M., PhD.

Implementación de métodos computacionales
Tecnológico de Monterrey

pperezm@tec.mx

04-2021

Contenido I

Introducción

Definición de sintaxis

- Componentes de una GIC

- Árbol de análisis

- Manejo de operadores

Análisis sintáctico

- Análisis de descenso recursivo

- Recursión por la izquierda

- Conjunto FIRST

Traducción dirigida por sintaxis

- Notación postfija

- Reglas semánticas

Introducción

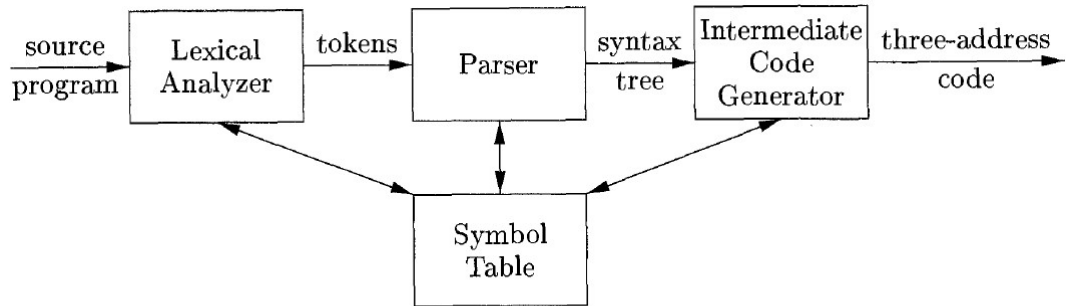
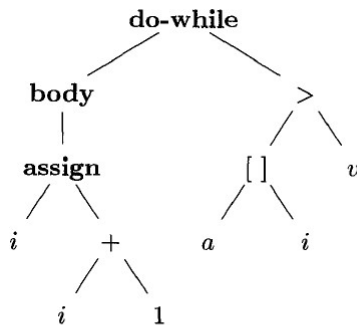


Figure 2.3: A model of a compiler front end



(a)

```
1: i = i + 1
2: t1 = a [ i ]
3: if t1 < v goto 1
```

(b)

Figure 2.4: Intermediate code for “do $i = i + 1$; while ($a[i] < v$);”

Componentes de una gramática

Una gramática libre de contexto tiene 4 componentes:

- ▶ Un conjunto de símbolos terminales llamados “tokens”.
- ▶ Un conjunto de símbolos no terminales, algunas veces llamados “variables sintática”.
- ▶ Un conjunto de producciones, donde cada producción consiste de un no terminales, llamado encabezado o lado izquierdo de la producción, una flecha y una secuencia de terminales y no-terminales, llamados cuerpo o lado derecho de la producción.
- ▶ La designación de uno de los no-terminales como símbolo de inicio.

$$list \rightarrow list + digit \quad (1)$$

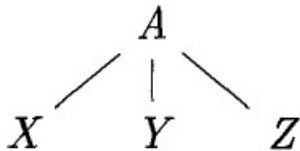
$$list \rightarrow list - digit \quad (2)$$

$$list \rightarrow digit \quad (3)$$

$$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \quad (4)$$

ϵ - cadena vacía

Árbol de análisis



- ▶ La raíz está etiquetada con el símbolo de inicio.
- ▶ Cada hoja está etiquetada con un terminal o ϵ .
- ▶ Cada nodo interior está etiquetado con un no terminal.
- ▶ Si A es la etiqueta de un nodo interno y X_1, X_2, \dots, X_n , entonces existe una producción $A \rightarrow X_1 X_2 \dots X_n$.

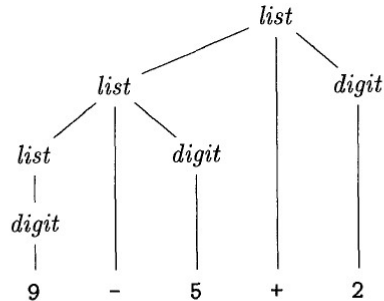


Figure 2.5: Parse tree for 9-5+2 according to the grammar in Example 2.1

$$\text{string} \rightarrow \text{string} + \text{string} \quad (5)$$

$$\text{string} \rightarrow \text{string} - \text{string} \quad (6)$$

$$\text{string} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \quad (7)$$

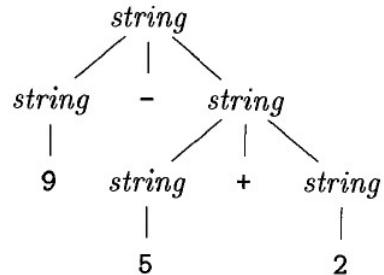
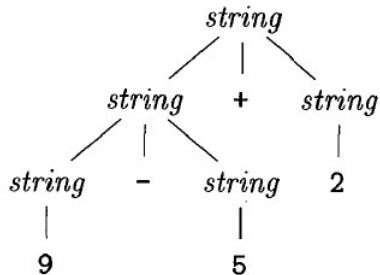


Figure 2.6: Two parse trees for 9-5+2

Asociatividad de operadores

$$list \rightarrow list + digit \quad (8)$$

$$list \rightarrow list - digit \quad (9)$$

$$list \rightarrow digit \quad (10)$$

$$list \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \quad (11)$$

$$right \rightarrow letter = right \quad (12)$$

$$right \rightarrow letter \quad (13)$$

$$letter \rightarrow a \mid b \mid \dots \mid z \quad (14)$$

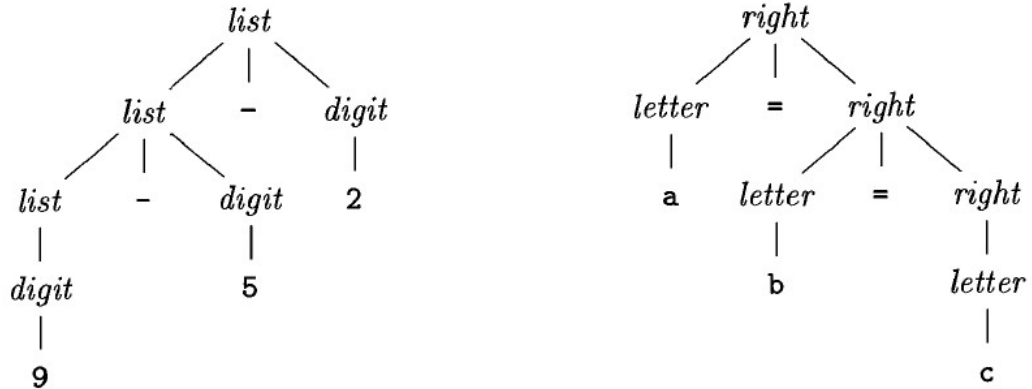


Figure 2.7: Parse trees for left- and right-associative grammars

Precedencia de operadores

$$expr \rightarrow expr + term \mid expr - term \mid term \quad (15)$$

$$term \rightarrow term * factor \mid term / factor \mid factor \quad (16)$$

$$factor \rightarrow digit \mid (expr) \quad (17)$$

En general los analizadores se pueden catalogar como: top-down (LL) o bottom-up (LR). Por ahora nos enfocaremos en **un analizador de descenso recursivo**.

- ▶ Es el más usado cuando realizamos un traductor simple dirigido por sintaxis.
- ▶ Aunque no es el más eficiente, ya que para cualquier gramática libre de contexto este analizador requiere un tiempo $O(n^3)$ para hacer su trabajo.

Análisis Top-Down

```

stmt  →  expr ;
        |  if ( expr ) stmt
        |  for ( optexpr ; optexpr ; optexpr ) stmt
        |  other

optexpr →   $\epsilon$ 
           |  expr
    
```

Figure 2.16: A grammar for some statements in C and Java

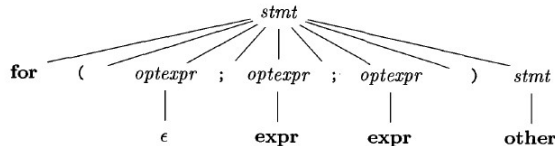


Figure 2.17: A parse tree according to the grammar in Fig. 2.16

Construcción top-down de un árbol de análisis:

- ▶ Para un node N , etiqueda con un no terminal A , selecciona una de las producciones existentes para A y genera los hijos de N .
- ▶ Encuentra el siguiente nodo cuyo subárbol no se haya construido.

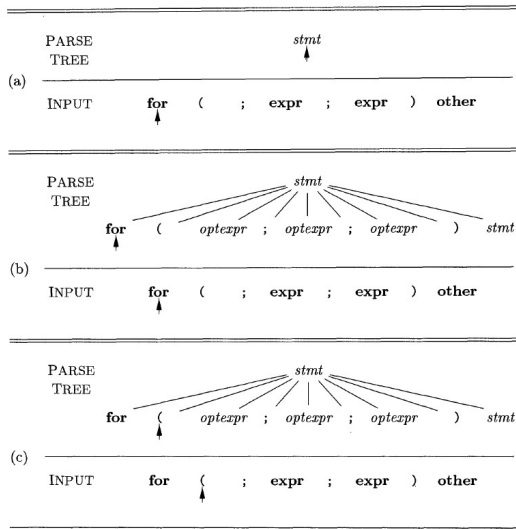


Figure 2.18: Top-down parsing while scanning the input from left to right

En general, la selección de una producción para un no terminal puede involucrar un proceso de prueba-y-error.

Análisis de descenso recursivo

Un analizador de descenso recursivo (ADR) emplea un método de análisis top-down en el cual un conjunto de procedimientos recursivos son usados para procesar la entrada. Existe un procedimiento asociado a cada no terminal de la gramática. En nuestro caso, revisaremos una forma simple de análisis de descenso recursivo llamado análisis predictivo, el cual emplea la lectura previa de un símbolo no ambiguo (lookahead) para determinar el flujo del proceso.


```
void stmt() {  
    switch ( lookahead ) {  
        case expr:  
            match(expr); match(';'); break;  
        case if:  
            match(if); match('('); match(expr); match(')'); stmt();  
            break;  
        case for:  
            match(for); match('(');  
            optexpr(); match(';'); optexpr(); match(';'); optexpr();  
            match(')'); stmt(); break;  
        case other:  
            match(other); break;  
        default:  
            report("syntax error");  
    }  
}  
  
void optexpr() {  
    if ( lookahead == expr ) match(expr);  
}  
  
void match(terminal t) {  
    if ( lookahead == t ) lookahead = nextTerminal;  
    else report("syntax error");  
}
```

Figure 2.19: Pseudocode for a predictive parser

El ADR se basa en la información obtenida de los primeros símbolos que genera una producción (FIRST).

$$\begin{array}{lcl} stmt & \rightarrow & \text{expr ;} \\ & | & \text{if (expr) stmt} \\ & | & \text{for (optexpr ; optexpr ; optexpr) stmt} \\ & | & \text{other} \\ \\ optexpr & \rightarrow & \epsilon \\ & | & \text{expr} \end{array}$$

Figure 2.16: A grammar for some statements in C and Java

- ▶ Un ADR utiliza una producción ϵ como acción por omisión cuando no existe una producción que pueda ser usada.
- ▶ Es importante mencionar que si queremos emplear este tipo de analizador, la gramática a reconocer debe tener conjuntos disjuntos FIRST para cualquier no terminal.

Como se mencionó anteriormente, un ADR es un programa que consiste de un procedimiento para cada no terminal. El procedimiento para el no terminal A hace dos cosas:

- ▶ Decide cual producción A usar examinando el símbolo lookahead. La producción con el cuerpo α es usado si el símbolo lookahead está en el conjunto $\text{FIRST}(\alpha)$.
- ▶ A continuación, el procedimiento reproduce el cuerpo de la producción elegida.

Recursión por la izquierda

La recursión por la izquierda es un problema grave para este tipo de analizadores, ya que los hace caer en un ciclo sin fin.

$$expr \rightarrow expr + term \quad (18)$$

Para resolver este problema ha que reescribir la producción. Si tenemos una producción del estilo:

$$A \rightarrow A\alpha|\beta \quad (19)$$

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & (E) \mid \mathbf{id} \end{array}$$

Reescribimos la producción A , usando una nueva no terminal R :

$$A \rightarrow \beta R \quad (20)$$

$$R \rightarrow \alpha R | \epsilon \quad (21)$$

Conjunto FIRST

- $\text{FIRST}(\alpha)$, donde α es cualquier cadena de símbolos de la gramática, es el conjunto de terminales que empieza una secuencia de cadenas que derivan de α .

$$\begin{array}{lcl} E & \rightarrow & T E' \\ E' & \rightarrow & + T E' \mid \epsilon \\ T & \rightarrow & F T' \\ T' & \rightarrow & * F T' \mid \epsilon \\ F & \rightarrow & (E) \mid \text{id} \end{array}$$

1. If X is a terminal, then $\text{FIRST}(X) = \{X\}$.
2. If X is a nonterminal and $X \rightarrow Y_1 Y_2 \cdots Y_k$ is a production for some $k \geq 1$, then place a in $\text{FIRST}(X)$ if for some i , a is in $\text{FIRST}(Y_i)$, and ϵ is in all of $\text{FIRST}(Y_1), \dots, \text{FIRST}(Y_{i-1})$; that is, $Y_1 \cdots Y_{i-1} \xRightarrow{*} \epsilon$. If ϵ is in $\text{FIRST}(Y_j)$ for all $j = 1, 2, \dots, k$, then add ϵ to $\text{FIRST}(X)$. For example, everything in $\text{FIRST}(Y_1)$ is surely in $\text{FIRST}(X)$. If Y_1 does not derive ϵ , then we add nothing more to $\text{FIRST}(X)$, but if $Y_1 \xRightarrow{*} \epsilon$, then we add $\text{FIRST}(Y_2)$, and so on.
3. If $X \rightarrow \epsilon$ is a production, then add ϵ to $\text{FIRST}(X)$.

Ver código fuente

Traducción dirigida por sintaxis es hecha al agregar reglas o fragmentos de código a las producciones de una gramática.

En este rubro, existen dos conceptos importantes:

- ▶ **Atributos.** Un atributo es cualquier cantidad asociada con una construcción de programación.
- ▶ **Esquemas de traducción dirigida por sintaxis.** Un esquema de traducción es una notación que agrega fragmentos de programa a las producciones una gramática.

Notación postfija para una expresión E se define:

- ▶ Si E es una variable o constante, entonces la notación postfija para E es la misma expresión E .
- ▶ Si E es una expresión de la forma $E_1 \text{ op } E_2$, donde **op** es un operador binaria, entonces la notación postfija para E es $E'_1 E'_2 \text{ op}$, donde E'_1 y E'_2 son las notaciones postfija para E_1 y E_2 , respectivamente.
- ▶ Si E es una expresión encerrada en paréntesis de la forma (E_1) , entonces la notación postfija es la misma que la notación postfija de E_1 .

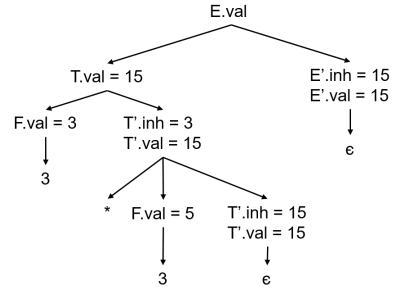
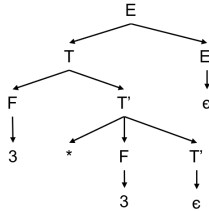
Asociamos atributos a los terminales y no terminales. Entonces, agregamos reglas de traducción a las producciones de la gramática. Estas reglas describen como los atributos son calculados en los nodos del árbol de análisis donde la producción.

EVALUANDO 3*5

ÁRBOL SINTÁCTICO

DEFINICIÓN DIRIGIDA POR SINTAXIS

PRODUCTION	SEMANTIC RULES
1) $T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \rightarrow * F T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$



Ver código fuente