

Reconocimiento Ascendente Predictive Parsing



Predictive Parsing

Predictive Parsing...

- Estrategia del algoritmo de parseo Top-Down llamada “predictive parsing”
- Similar a “recursive descendant” pero el parseador puede “predecir” cual producción utilizar.
 - Mirando algunos tokens siguientes (look-ahead) y restringe la gramática
 - Sin backtracking (parseador totalmente determinístico)
- Predictive Parsers aceptan gramáticas LL(K)

Left-to-right

Left-most derivation

K tokens look-ahead (K=1)

Predictive Parsing...

- En Recursive Descendent
 - En cada paso, muchas opciones de producciones a utilizar
 - Backtracking es usado para deshacer incorrectas selecciones
- Parser LL(1)
 - En cada paso, solamente una opción de producción a utilizar

Predictive Parsing...

- Usando la gramática $E \rightarrow T + E \mid T$
 $T \rightarrow \text{int} \mid \text{int}^* T \mid (E)$
The two "int" terms in the second production are highlighted with yellow boxes.
- Es difícil predecir debido a
 - Hay dos producciones en T que inician con “int”
 - Para E no esta claro la forma de predecir
- Esta gramática es inaceptable para el parseo predictivo (al menos para LL1)
- Es necesaria la aplicación de “**left-factoring**” sobre la gramática
 - Eliminación de prefijos comunes de múltiples producciones para un NoTerminal

Predictive Parsing...

- Usando la gramática

$$\begin{array}{l} E \rightarrow T + E \mid T \\ T \rightarrow \text{int} \mid \text{int} * T \mid (E) \end{array}$$

Inician con el mismo prefijo

- Se retardará la decisión sobre la producción a utilizar

$$\begin{array}{l} E \rightarrow T X \\ X \rightarrow + E \mid \epsilon \end{array}$$

$$\begin{array}{l} T \rightarrow \text{int} Y \mid (E) \\ Y \rightarrow * T \mid \epsilon \end{array}$$

Predictive Parsing...

- Gramática Left-factored

$$E \rightarrow TX$$

$$X \rightarrow + E \mid \epsilon$$

$$T \rightarrow (E) \mid \text{int } Y$$

$$Y \rightarrow * T \mid \epsilon$$

- Se utilizará para construir la tabla de parseo (LL(1) parsing table)

		Siguiente token de entrada					
		int	*	+	()	\$
E	TX				TX		
X				+ E		ϵ	ϵ
T	int Y				(E)		
Y		* T		ϵ		ϵ	ϵ

 No Terminal LeftMost

 la entrada es el lado derecho de la producción de usar

Predictive Parsing...

- Considere la entrada [E, int]
 - Cuando el NoTerminal actual es E y la siguiente entrada es int, use la producción

$$E \rightarrow T X$$

	int	*	+	()	\$
E	TX			TX		
X			+ E		ϵ	ϵ
T	int Y			(E)		
Y		* T	ϵ		ϵ	ϵ

Predictive Parsing...

- Considere la entrada $[Y, +]$
 - Cuando el NoTerminal actual es Y y la siguiente entrada es $+$, deshacerse de Y
 - Y puede ser seguido por $+$ solamente si $Y \rightarrow \epsilon$

	int	*	+	()	\$
E	TX			TX		
X			+ E		ϵ	ϵ
T	int Y			(E)		
Y		* T	ϵ		ϵ	ϵ

Predictive Parsing...

- Considere la entrada [E, *]
 - No hay una forma de derivar una hilera iniciando con * desde el NoTerminal E

Observe que muchas entradas son blancos y representan entradas de error.

	int	*	+	()	\$
E	TX			TX		
X			+ E		ϵ	ϵ
T	int Y			(E)		
Y		* T	ϵ		ϵ	ϵ

Predictive Parsing...

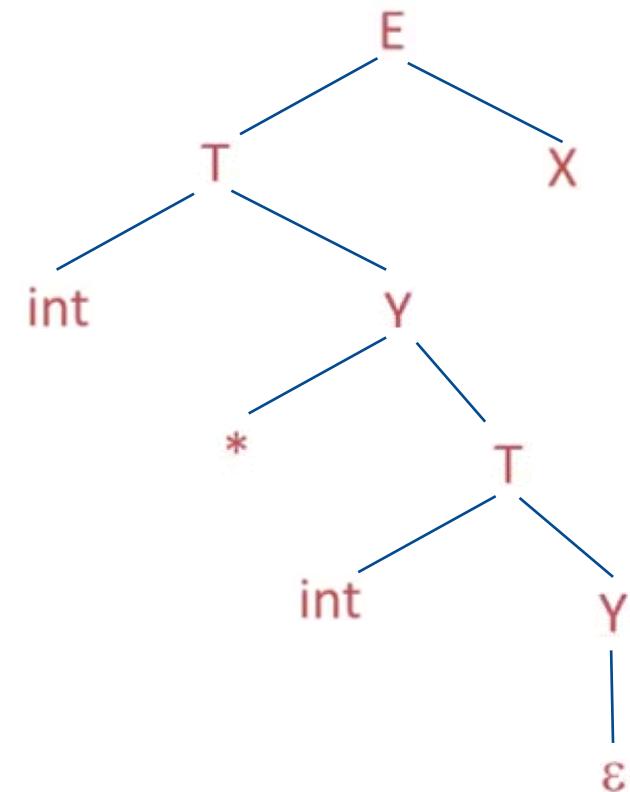
- El método usado para parsear usando la tabla de parseo es similar al método de recursive descendant, excepto por:
 - Para el NoTerminal LeftMost S
 - Se mirará al siguiente token de entrada a
 - Se selecciona la producción mostrada en $[S,a]$
- En lugar de funciones de recursividad se tiene una pila de registros que pueden registrar el avance del árbol de parseo
 - Ningún punto en el árbol tendrá algún NoTerminal que aún no se haya extendido
 - Terminales que aún no han sido emparejados con la entrada
 - Top de la pila: terminales leftmost pendientes o NoTerminales
- Rechazar al llegar al estado de error
- Aceptar el final de la entrada y pila vacía

Predictive Parsing...

```
initialize stack = <S $> and next
repeat
    case stack of
        <X, rest> : if T[X,*next] = Y1...Yn
                      then stack ← <Y1... Yn rest>;
                      else error ();
        <t, rest>   : if t == *next ++
                      then stack ← <rest>;
                      else error ();
    until stack == <>
```

Predictive Parsing...

Stack	Input	Action
E \$	int * int \$	T X
TX \$	int * int \$	int Y
int Y X \$	int * int \$	terminal
Y X \$	* int \$	* T
* TX \$	* int \$	terminal
TX \$	int \$	int Y
int Y X \$	int \$	terminal
Y X \$	\$	ϵ
X \$	\$	ϵ



Predictive Parsing...

LL(1)

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow + TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow * FT' \mid \epsilon \\ F &\rightarrow (E) \mid id \end{aligned}$$

Tabla M :

N \ T	id	+	*	()	\$
E	TE'			TE'		
E'		+TE'			ϵ	ϵ
T	FT'			FT'		
T'		ϵ	*FT'		ϵ	ϵ
F	id			(E)		

Reconocer : id * (id + id) \$

Reconocer (id + id)(id + id) * id+ id \$

Predictive Parsing...

reconocer $(x; (x))$ para

$$\begin{array}{l} S \rightarrow (A) \\ A \rightarrow CB \\ B \rightarrow ; A \mid \epsilon \\ C \rightarrow x \mid S \end{array}$$

	()	;	x	\$
S	S \rightarrow (A)				
A	A \rightarrow CB				
B		B \rightarrow ϵ		B \rightarrow ; A	
C	C \rightarrow S				C \rightarrow x

Predictive Parsing...

pila	entrada	producción usada (salida)
\$S	(x; (x))\$	S → (A)
\$)A((x; (x))\$	
\$)A	x; (x))\$	A → CB
\$)BC	x; (x))\$	C → x
\$)Bx	x; (x))\$	
\$)B	; (x))\$	B → ; A
\$)A;	; (x))\$	
\$)A	(x))\$	A → CB
\$)BC	(x))\$	C → S
\$)BS	(x))\$	S → (A)
\$)B)A((x))\$	
\$)B)A	x))\$	A → CB
\$)B)BC	x))\$	C → x
\$)B)Bx	x))\$	
\$)B)B)\$	B → ε
\$)B))\$	
\$)B)\$	B → ε
\$))\$	
\$	\$	

First Set

Tablas de parseo LL1

First Sets...

- Considera un NoTerminal A , una producción $A \rightarrow \alpha$, & Token t
- $T[A,t] = \alpha$ en dos casos
- Si $\alpha \rightarrow^* t \beta$
 - α puede derivar una t en la primera posición
 - Se puede decir que $t \in \text{First}(\alpha)$
- Si $A \rightarrow \alpha$ y $\alpha \rightarrow^* \varepsilon$ y $S \rightarrow^* \beta A t \delta$
 - Útil si la pila tiene A , la entrada es t , y A no puede derivar a t
 - En este caso, la única opción es deshacerse de A (por derivación ε)
 - Puede trabajar solamente si t puede seguir a A en la última derivación
 - Se puede decir $t \in \text{Follow}(A)$
- A no produce t... t aparece en una derivación “después” de A

$\alpha \not\rightarrow^* t$

$t \notin \text{First}(\alpha)$

First Sets...

- Definición

$$\text{First}(X) = \{ t \mid X \rightarrow^* t\alpha \} \cup \{ \varepsilon \mid X \rightarrow^* \varepsilon \}$$

- X es un string. X puede derivar a t en la primer posición

- Algoritmo:

1. $\text{First}(t) = \{ t \}$ donde t es un terminal. Cualquier símbolo terminal, es en sí, el primer conjunto que consiste en solo ese terminal.

2. $\varepsilon \in \text{First}(X)$ donde X es un NoTerminal.

- Si $X \rightarrow \varepsilon$ lo cual significa, que ε (épsilon) debe ser el First de X .
- Si $X \rightarrow A_1 \dots A_n$ y $\varepsilon \in \text{First}(A_i)$ para $1 \leq i \leq n$. Incluso aunque X pueda tener otras producciones, épsilon será el First.
- Significa que épsilon es el first de todos los NoTerminales

3. $\text{First}(\alpha) \subseteq \text{First}(X)$ si $X \rightarrow A_1 \dots A_n \alpha$ y $\varepsilon \in \text{First}(A_i)$ para $1 \leq i \leq n$.

- Cualquier cosa que sea el First de alpha será también el First de X

First Sets...

- La siguiente gramática

$$\begin{array}{l} E \rightarrow T\ X \\ T \rightarrow (\ E) \mid \text{int}\ Y \end{array}$$

$$\begin{array}{l} X \rightarrow +\ E \mid \varepsilon \\ Y \rightarrow *\ T \mid \varepsilon \end{array}$$

Terminales

$$\text{First}(+) = \{ + \}$$

$$\text{First}(*) = \{ * \}$$

$$\text{First}(()) = \{ () \}$$

$$\text{First}()) = \{) \}$$

$$\text{First}(\text{int}) = \{ \text{int} \}$$

NoTerminales

$$\text{First}(E) \geq \text{First}(T)$$

$$\text{First}(T) = \{ (, \text{int} \}$$

$$\text{First}(X) = \{ +, \varepsilon \}$$

$$\text{First}(Y) = \{ *, \varepsilon \}$$

Follow Sets

Tablas de parseo LL1

Follow Sets...

- Definición

$$\text{Follow}(X) = \{ t \mid S \xrightarrow{*} \beta X t \delta \}$$

Follow set trata sobre el símbolo que se puede generar, sino de todos los símbolos que puede generar.

T es el Follow(X) si dentro de la gramática, alguna derivación donde t aparece inmediatamente después de X.

- Intuición

- Si $X \rightarrow A B$ entonces $\text{First}(B) \subseteq \text{Follow}(A)$ y
 $\text{Follow}(X) \subseteq \text{Follow}(B)$

- Si $B \xrightarrow{*} \epsilon$ entonces $\text{Follow}(X) \subseteq \text{Follow}(A)$

- Si S es el símbolo inicial entonces $\$ \in \text{Follow}(S)$

Follow Sets...

- Algoritmo

1. $\$ \in \text{Follow}(S)$

2. $\text{First}(\beta) - \{\varepsilon\} \subseteq \text{Follow}(X)$

- Para cada producción $A \rightarrow \alpha X \beta$

3. $\text{Follow}(A) \subseteq \text{Follow}(X)$

- Para cada aplicación $A \rightarrow \alpha X \beta$ donde $\varepsilon \in \text{First}(\beta)$

Follow Sets...

No interesa incluir épsilon en el Follow

- La siguiente gramática

$$\begin{aligned} E &\rightarrow T \ X \\ T &\rightarrow (\ E) \mid \text{int} \ Y \end{aligned}$$

$$\begin{aligned} X &\rightarrow + \ E \mid \epsilon \\ Y &\rightarrow * \ T \mid \epsilon \end{aligned}$$

Subconjuntos

$$\begin{aligned} \text{Follow}(X) &\subseteq \text{Follow}(E) \\ \text{Follow}(E) &= \text{Follow}(X) \\ \text{Follow}(E) &\cap \text{Follow}(T) \\ \text{Follow}(Y) &= \text{Follow}(T) \\ \text{Follow}(T) &= \text{Follow}(Y) \end{aligned}$$

Follow's

$$\begin{aligned} \text{FOLLOW}(E) &= \{ \$,) \} \\ \text{FOLLOW}(X) &= \{ \$,) \} \\ \text{FOLLOW}(T) &= \{ +, \$,) \} \\ \text{FOLLOW}(Y) &= \{ +, \$,) \} \\ \text{FOLLOW}('(') &= \{ (, \text{int} \} \\ \text{FOLLOW}(')') &= \{ +, \$,) \} \\ \text{FOLLOW}('+') &= \{ (, \text{int} \} \\ \text{FOLLOW}('*') &= \{ (, \text{int} \} \\ \text{FOLLOW}(\text{int}) &= \{ *, +, \$,) \} \end{aligned}$$

Tablas de parseo LL1

Tablas de parseo LL(1)...

- Construir una tabla de parseo T para una gramática libre de contexto (CFG) G
- Para cada producción $A \rightarrow \alpha$ en G hacer:
 - Para cada terminal $t \in \text{First}(\alpha)$ hacer
 - $T[A, t] = \alpha$
 - Si $\epsilon \in \text{First}(\alpha)$ para cada $t \in \text{Follow}(A)$ hacer
 - $T[A, t] = \alpha$
 - Si $\epsilon \in \text{First}(\alpha)$ y $\$ \in \text{Follow}(A)$ hacer
 - $T[A, \$] = \alpha$

Tablas de parseo LL(1)...

- La siguiente gramática

$$\begin{array}{l} E \rightarrow T X \\ T \rightarrow (E) \mid \text{int } Y \end{array}$$

$$\begin{array}{l} X \rightarrow + E \mid \varepsilon \\ Y \rightarrow * T \mid \varepsilon \end{array}$$

$$\begin{array}{l} \text{Follow}(X) = \{ \$,) \} \\ \text{Follow}(Y) = \{ +, \$,) \} \end{array}$$

	()	+	*	int	\$
E	TX				TX	
T	(E)				int Y	
X	Error	ε	+E			ε
Y	...	ε	ε	*T		ε

Tablas de parseo LL(1)...

- Dada una gramática recursiva a la izquierda (left recursive)

$$S \rightarrow S a \mid b$$

$$\text{First}(S) = \{ b \}$$

$$\text{Follow}(S) = \{ \$, a \}$$

	a	b	\$
S		b Sa	



Una entrada que posee múltiples movimientos
Multiply defined

Significa que si se ve una S y se desea expandir, no se conoce cual movimiento hacer exactamente, debido a que no es determinístico.

Tablas de parseo LL(1)...

- Si alguna entrada es **Multiply defined** entonces G no es LL(1)
 - No left factored
 - Left recursive
 - Gramáticas ambiguas
 - Otras gramáticas que no posee un valor único por entrada.
- La mayoría de los lenguajes de programación CFGs no son LL(1)

Tablas de parseo LL(1)...

- Ejercicio
- Genere la tabla LL(1) para la siguiente gramática

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

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

- Reconozca la hilera

$$a^* (b + c) \equiv id^* (id + id)$$

Tablas de parseo LL(1)...

- Ejercicio
- Genere la tabla LL(1) para la siguiente gramática

$E \rightarrow T E'$
 $E' \rightarrow + T E' \mid \epsilon$
 $T \rightarrow F T'$
 $T' \rightarrow * F T' \mid \epsilon$
 $F \rightarrow (E) \mid id$

$$a^* (b + c) \equiv id^* (id + id)$$

- Tabla

N \ T	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Reconocimiento Ascendente Predictive Parsing

