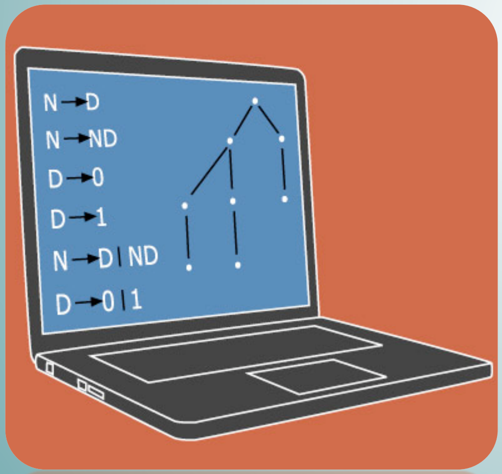
Procesadores de Lenguaje

Analizadores sintácticos descendentes: LL(1)



Cristina Tirnauca

DPTO. DE MATEMÁTICAS,

ESTADÍSTICA Y COMPUTACIÓN

Analizadores sintácticos

Los analizadores descendentes:

- Corresponden a un autómata con pila determinista.
- Construyen un árbol sintáctico de la raíz hacia las hojas (del símbolo inicial hacia los símbolos terminales).
- Por ejemplo: el analizador LL o predictivo lee los datos de izquierda a derecha (*Left to right*) y construye la derivación izquierda (*Leftmost*).
- Emplea una pila para mantener un resumen de lo que espera ver a continuación hasta el final de los datos.
- La recursividad izquierda les puede causar problemas.

Los analizadores ascendentes (shift-reduce):

- Corresponden a un autómata con pila determinista.
- Construyen un árbol sintáctico de las hojas hacia la raíz (de los terminales hacia el símbolo inicial de la gramática).
- Por ejemplo: los analizadores LR leen los datos de izquierda a derecha (*Left to right*) y construyen la derivación derecha (*Rightmost*). ("al revés")
- Emplean una pila para mantener un resumen de lo que llevan visto hasta el momento.
- Son más eficientes con recursividad izquierda.

Planteamiento

Suponemos que un analizador léxico nos proporciona el siguiente *token* cuando pedimos leer uno, o un *token* “nulo” (representado por $) en caso de fin de datos.

(Algunas referencias incluyen también un nuevo símbolo inicial que deriva en el símbolo inicial seguido del marcador de fin de datos:
S’ $\rightarrow$ S $.)

Idea clave:

La entrada se deriva del símbolo inicial si y sólo si lo que queda por leer se deriva de lo que hay en la pila.

Herramientas conceptuales auxiliares

Tres nociones importantes

Sobre palabras formadas por símbolos de la gramática, tanto terminales como no terminales.

- Anulabilidad de una palabra,
- FIRST de una palabra (terminales por los que puede empezar una parte de la entrada que derive de esa palabra),
- FOLLOW de una palabra (terminales que pueden aparecer en una entrada válida justo a continuación de una parte de la entrada que derive de esa palabra).

Anulabilidad

Es decir, capacidad para "desaparecer"

Una palabra es anulable si puede derivar en la palabra vacía.

- Palabras que contienen algún símbolo terminal:

  Nunca pueden derivar en la palabra vacía, porque un símbolo terminal que participa en una derivación ya no puede desaparecer de ella.

- Palabras que sólo contienen símbolos no terminales:
  - λ es anulable;
  - si α y β son anulables, $\alpha\beta$ son anulables;
  - si la gramática contiene una regla $X \to \alpha$ y α es anulable, X es anulable.

Lo calculamos de manera iterativa, alternativamente para símbolos no terminales y para partes derechas de las reglas: nada es anulable hasta que se demuestra lo contrario.

FIRST

$\text{FIRST}(\alpha) = \{a \mid \alpha \text{ puede derivar en } a\gamma\}$, donde a es un símbolo terminal y α y γ son palabras formadas por símbolos terminales o no terminales.

FIRST(α): conjuntos lo más pequeños posible tales que

- $\text{FIRST}(a) = \{a\}$ para cada símbolo terminal a;
- si $\alpha = X\beta$ y X no es anulable, $\text{FIRST}(\alpha) = \text{FIRST}(X)$;
- si $\alpha = X\beta$ y X es anulable, $\text{FIRST}(\alpha) = \text{FIRST}(X) \cup \text{FIRST}(\beta)$;
- si la gramática contiene una regla $X \to \alpha$, FIRST(X) incluye FIRST(α).

Cálculo iterativo: inicialmente no sabemos de ningún terminal y empezamos por $\emptyset$; y alternamos entre símbolos no terminales y partes derechas de reglas.

FOLLOW

$\text{FOLLOW}(X) = \{a \mid S \text{ puede derivar en } \alpha X a \gamma\}$, donde X es un símbolo no terminal, a es un símbolo terminal, α y γ son palabras formadas por símbolos terminales o no terminales, y S es el símbolo inicial de la gramática.

FOLLOW(X): conjuntos lo más pequeños posible tales que

- FOLLOW(S) incluye el fin de datos (representado aquí \$);
- si la gramática contiene una regla $X \to Y_1 \dots Y_k$ y $Y_{i+1} \dots Y_{j-1}$ es anulable, con $1 \leq i < j \leq k$, FOLLOW(Y_i) incluye FIRST(Y_j) (prestemos atención al caso $i + 1 = j$).
- si la gramática contiene una regla $X \to Y_1 \dots Y_k$ y $Y_{i+1} \dots Y_k$ es anulable, con $1 \leq i \leq k$, FOLLOW(Y_i) incluye FOLLOW(X) (prestemos atención al caso $i = k$).

Un ejemplo (Anulabilidad, FIRST y FOLLOW)

Consideremos $G = (\{E, E', T, T', F\}, \{+, *, (,), id\}, E, P)$, donde P consiste en las reglas siguientes:

$$E \rightarrow TE' \qquad T \rightarrow FT' \qquad F \rightarrow (E) \mid id$$
$$E' \rightarrow +TE' \mid \lambda \qquad T' \rightarrow *FT' \mid \lambda$$

	Anulable	FIRST	FOLLOW
E	no	$(, id$	$\$,)$
E'	sí	$+$	$\$,)$
T	no	$(, id$	$\$,), +$
T'	sí	$*$	$\$,), +$
F	no	$(, id$	$\$,), +, *$

Gramáticas y lenguajes LL(1)

Definición

Una gramática libre de contexto $G = (V, \Sigma, S, P)$ es $LL(1)$ si

- $S \Rightarrow^*_{lm} wX\gamma \Rightarrow w\alpha\gamma \Rightarrow^*_{lm} wu$,
- $S \Rightarrow^*_{lm} wX\delta \Rightarrow w\beta\delta \Rightarrow^*_{lm} wv$,
- $FIRST(u) = FIRST(v)$

implican $\alpha = \beta$, donde $u, v, w \in \Sigma^*$ y $X \in V$.

Se dice sobre un lenguaje que es LL(1) si se puede generar con una gramática LL(1).

Ejemplos

Gramáticas LL(1)

- $G = (\{S\}, \{(,)\}, S, P = \{S \rightarrow (S)|\lambda\})$
  En esta gramática es obvio que la primera producción se usa cuando aparece un paréntesis abierto y la segunda cuando aparece el primer paréntesis cerrado.
- $G = (\{S\}, \{a, b\}, S, P = \{S \rightarrow aAb|b, A \rightarrow aSAa|b\})$

Una gramática LL(2) que no es LL(1)

$G = (\{S\}, \{a, b\}, S, P = \{S \rightarrow abSba|aa\})$

Ejemplo de lenguaje que no es LL(k) para ningún k

$L = \{a^n cb^n \mid n \geq 1\} \cup \{a^n db^{2n} \mid n \geq 1\}$

El analizador LL descendente

El analizador LL está utilizando:

- un "buffer" para la entrada
- una pila, con símbolos terminales y no terminales
- una tabla de análisis

En cada paso, el analizador lee un símbolo del buffer y el símbolo que está en la cima de la pila.

- si coinciden, el analizador los elimina del buffer y de la pila
- si en la cima de la pila hay un símbolo terminal distinto, devuelve ERROR (la palabra no está aceptada)
- si en la cima de la pila hay un símbolo no terminal X, el analizador "mira" la tabla para ver que regla se debe aplicar, y substituye X por la parte derecha de esa regla.

Al principio la pila contiene el símbolo inicial S y el símbolo especial \$ (el fondo de la pila).

Ejemplo

$G = (\{S, F\}, \{a, (,)\}, S, P)$ con P dado por las reglas siguientes:

1. $S \rightarrow F$
2. $S \rightarrow (S + F)$
3. $F \rightarrow a$

Table: FIRST y FOLLOW

	Anulable	FIRST	FOLLOW
S	no	$a, ($	$+, \$$
F	no	a	$+,), \$$

Table: Tabla de análisis

	(	)	a	$+$	\$
S	2	-	1	-	-
F	-	-	3	-	-

Entrada: $w = (a + a)$

Tabla de análisis

Para cada símbolo no terminal X y cada símbolo terminal a, añadimos la regla $X \to \alpha$ si:

- $a \in \text{FIRST}(\alpha)$, o
- α es anulable y $a \in \text{FOLLOW}(X)$

Si la tabla contiene a lo sumo una regla para cada una de sus celdas, entonces el analizador siempre sabe que regla se debe utilizar en cada momento.

Conflictos

- FIRST/FIRST conflict (dos alternativas empiezan igual)

$$A \rightarrow X\alpha \mid X\beta$$

$$A \rightarrow XB$$
$$B \rightarrow \alpha \mid \beta$$

- FIRST/FOLLOW conflict (el FIRST y el FOLLOW de un símbolo no terminal anulable tienen algo en común)

$$S \rightarrow Aab$$
$$A \rightarrow a \mid \lambda$$

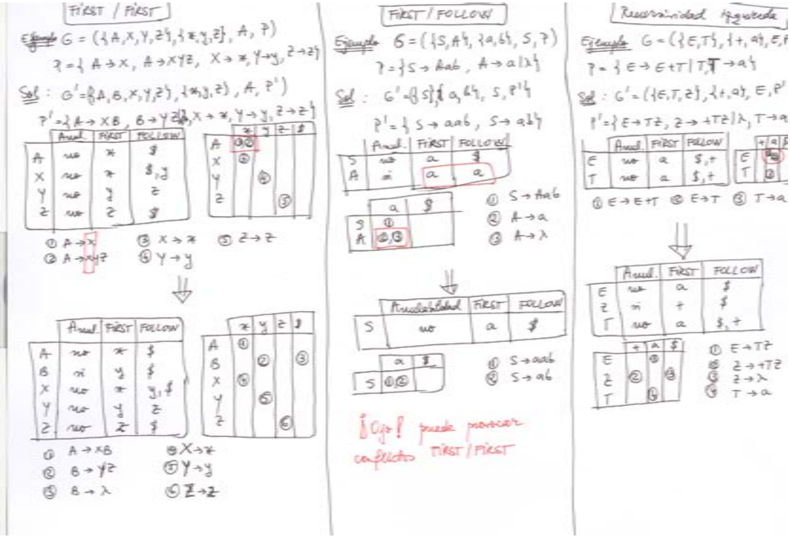
$$S \rightarrow aab \mid ab$$

- Recursividad izquierda

$$E \rightarrow E + T \mid T$$

$$E \rightarrow TZ$$
$$Z \rightarrow +TZ \mid \lambda$$

Conflictos (II)



Recursividad Izquierda

La bendición del análisis ascendente

La maldición del análisis descendente: el recursivo entra en bucle.
Garantiza ambigüedad en la tabla LL(1).
Frecuentemente se puede resolver el problema cambiando un poco la gramática:

E $\rightarrow$ E α_1 | E α_2 | ... | E α_k
E $\rightarrow$ β_1 | β_2 | ... | β_m

(donde las β no pueden derivar en nada que empiece por E)

E es: $(\beta_1|\beta_2|\ldots|\beta_m)(\alpha_1|\alpha_2|\ldots|\alpha_k)$*

Z $\rightarrow$ α_1 Z | α_2 Z | ... | α_k Z | λ
E $\rightarrow$ β_1 Z | β_2 Z | ... | β_m Z

Práctica 2

Fecha límite entrega: miercoles, 7 de marzo de 2012, a las 8:30 h
Forma de entrega: en papel (en persona) o .pdf por correo electrónico dirigido a cristina.tirnauca@unican.es

Problema 1: Recursividad Izquierda

E $\rightarrow$ E OP E | num

OP $\rightarrow$ '+' | '*'

(Construir la tabla. Modificar la gramática. Construir la tabla.)

Problema 2: Factorización Izquierda (no siempre funciona)

S $\rightarrow$ if expr then S else S
S $\rightarrow$ if expr then S
S $\rightarrow$ asign
(Construir la tabla. Modificar la gramática. Construir la tabla. Identificar el tipo de conflicto. Aplicar la solución propuesta en clase. ¿Con qué grámatica nos encontramos?)