Tema 3: Análisis sintáctico descendente

Procesamiento de Lenguajes

Dept. de Lenguajes y Sistemas Informáticos
Universidad de Alicante

¿Cómo funciona el analizador sintáctico?

- La primera acción de un analizador sintáctico es obtener un token de la entrada, llamando al analizador léxico (que trabaja como un subprograma)
- El analizador va leyendo tokens del analizador léxico a la vez que va generando la traducción, comprobando que la sintaxis es correcta y comprobando las restricciones semánticas.
- IMPORTANTE: Las tres tareas (traducción, sintaxis, semántica) se realizan de forma simultánea, aunque a veces es necesario acumular varios tokens para realizar alguna comprobación o generar la traducción

Ejemplo: gramática de expresiones simples

$$
\begin{array}{lcl}
E & \rightarrow & E \ \textbf{opsuma} \ T \\
E & \rightarrow & T \\
T & \rightarrow & T \ \textbf{opmul} \ F \\
T & \rightarrow & F \\
F & \rightarrow & \textbf{id} \\
F & \rightarrow & \textbf{num} \\
F & \rightarrow & \textbf{pari} \ E \ \textbf{pard}
\end{array}
$$

`2+3*4`	`suma(2,prod(3,4))`
`2+3-4`	`resta(suma(2,3),4)`
`2+3*(4-5)`	`suma(2,prod(3,resta(4,5)))`

Algoritmos de análisis sintáctico

- Para cualquier GIC: Cocke-Younger-Kasami (CYK), Earley, Tomita, ... $\approx O(n^3)$
- Si se desea un coste temporal lineal, $O(n)$, es necesario poner restricciones a las GIC, es decir, usar subconjuntos del conjunto de las GIC. Hay dos estrategias:
  - Análisis sintáctico descendente (ASD)
  - Análisis sintáctico ascendente (ASA)
- Implementación de analizadores sintácticos:
  - *A mano* (para gramáticas simples)
  - Usando generadores automáticos: yacc/bison, ANTLR, PCCTS, ...

Análisis sintáctico descendente: un ejemplo

El análisis sintáctico descendente trata de reproducir la derivación por la izquierda de la cadena de entrada (solo puede haber una)
Ejemplo: **int a,b,c;**

$$
\begin{array}{lcl}
D & \rightarrow & \textit{Tipo}\ \textbf{id}\ L \\
\textit{Tipo} & \rightarrow & \textbf{int} \\
\textit{Tipo} & \rightarrow & \textbf{float} \\
L & \rightarrow & \textbf{coma}\ \textbf{id}\ L \\
L & \rightarrow & \textbf{pyc}
\end{array}
$$

$$
\begin{array}{lcl}
D & \Rightarrow & \textit{Tipo}\ \textbf{id(a)}\ L \\
 & \Rightarrow & \textbf{int}\ \textbf{id(a)}\ L \\
 & \Rightarrow & \textbf{int}\ \textbf{id(a)}\ \textbf{coma}\ \textbf{id(b)}\ L \\
 & \Rightarrow & \textbf{int}\ \textbf{id(a)}\ \textbf{coma}\ \textbf{id(b)}\ \textbf{coma}\ \textbf{id(c)}\ L \\
 & \Rightarrow & \textbf{int}\ \textbf{id(a)}\ \textbf{coma}\ \textbf{id(b)}\ \textbf{coma}\ \textbf{id(c)}\ \textbf{pyc}
\end{array}
$$

Análisis sintáctico ascendente: un ejemplo

El análisis sintáctico ascendente trata de reconstruir la inversa de una derivación por la derecha de la cadena de entrada
Ejemplo:

int id(a) coma id(b) coma id(c) pyc	$\Leftarrow$
Tipo **id(a) coma id(b) coma id(c) pyc**	$\Leftarrow$
Tipo **id(a) coma id(b) coma id(c)** *L*	$\Leftarrow$
Tipo **id(a) coma id(b)** *L*	$\Leftarrow$
Tipo **id(a)** *L*	$\Leftarrow$
D	

Características no deseables para el análisis lineal

- Análisis sintáctico descendente
  - Recursividad por la izquierda
  - Factores comunes por la izquierda
  - Ambigüedad
- Análisis sintáctico ascendente
  - Ambigüedad

Análisis sintáctico descendente (ASD)

- Se llama *descendente* porque parte del símbolo inicial (la raíz del árbol de derivación) y trata de llegar a la cadena de terminales (las hojas del árbol)
- Para que se pueda usar en un compilador, el analizador sintáctico debe tener un coste temporal lineal, $O(n)$
- Trata de, leyendo la cadena de entrada de izquierda a derecha (*left-to-right*), obtener la derivación válida por la izquierda de la cadena de entrada (*leftmost derivation*).

Condiciones para el ASD en tiempo lineal

- Para realizar un análisis sintáctico lineal, el analizador debe saber en todo momento qué regla ha de aplicar, no puede hacer *backtracking*. Por tanto, debe ser un ASD *predictivo*
- Siempre se tiene que tratar de derivar el no terminal más a la izquierda en la cadena de símbolos. Inicialmente la cadena de símbolos sólo contiene el símbolo inicial, pero según se van aplicando reglas contiene terminales y no terminales.
  Ejemplo:

$$
\begin{array}{lcl}
A & \longrightarrow & \mathbf{a}\ B\ C \\
B & \longrightarrow & \mathbf{b}\ \mathbf{bas} \\
B & \longrightarrow & \mathbf{big}\ C\ \mathbf{boss} \\
C & \longrightarrow & \epsilon \\
C & \longrightarrow & \mathbf{c}
\end{array}
$$

$$
A \Rightarrow \mathbf{a}\ B\ C \Rightarrow \mathbf{a}\ \mathbf{b}\ \mathbf{bas}\ C\ \ldots
$$

Condiciones para el ASD predictivo

- A la vez que se van aplicando reglas, hay que comprobar que los terminales que aparecen (por la izquierda) coinciden (*match*) con los que aparecen en la cadena de entrada.
- Pero, dado un no terminal B , ... ¿cómo se puede predecir qué regla hay que aplicar? *Mirando los primeros símbolos de las partes derechas de* B
  Ejemplo:

Gramática			Entrada	Derivación
A	$\longrightarrow$	**a** B C	**a b bas c**	A
B	$\longrightarrow$	**b bas**	**a b bas c**	**a** B C
B	$\longrightarrow$	**big** C **boss**	**b bas c**	B C
C	$\longrightarrow$	ϵ	**b bas c**	**b bas** C
C	$\longrightarrow$	**c**	**bas c**	**bas** C
			c	C
			c	**c**
			$	**$**

Algoritmo de análisis sintáctico descendente predictivo

1. Inicialmente se tiene la cadena de entrada y el símbolo inicial de la gramática en la derivación
2. Repetir hasta llegar al final de la cadena de entrada (**$**):
   - Si el símbolo más a la izquierda en la derivación es un no terminal, predecir qué regla aplicar en función del símbolo que hay en la entrada, y aplicar la regla ¿Y si no hay ninguna regla aplicable?
   - Si ese símbolo es un terminal, hay que compararlo con el símbolo de la entrada y avanzar ¿Y si no coinciden?

Predicción de la regla a aplicar (1)

Para elegir (predecir) qué regla aplicar para un no terminal A, hay que consultar la parte derecha de las reglas de A:

$$
\begin{array}{lcl}
A & \longrightarrow & \textbf{all}\ B\ C \\
A & \longrightarrow & \textbf{bad} \\
B & \longrightarrow & \textbf{big}\ C\ \textbf{boss} \\
B & \longrightarrow & \textbf{bet} \\
C & \longrightarrow & \textbf{cat} \\
C & \longrightarrow & \textbf{cow}
\end{array}
$$

En este caso, está *chupao*, basta con mirar el terminal de la entrada y el no terminal a derivar, y se elige la regla a aplicar

Predicción de la regla a aplicar (2)

... pero ¿y si al principio de la parte derecha hay un no terminal?

A	$\longrightarrow$	$B\ C$	$\{$ **big** , **bet** $\}$
A	$\longrightarrow$	**bad**	$\{$ **bad** $\}$
B	$\longrightarrow$	**big** C **boss**	$\{$ **big** $\}$
B	$\longrightarrow$	**bet**	$\{$ **bet** $\}$
C	$\longrightarrow$	**cat**	$\{$ **cat** $\}$
C	$\longrightarrow$	**cow**	$\{$ **cow** $\}$

En este caso, es necesario tener calculados el conjunto de terminales que aparecen al principio de la parte derecha de las reglas de un no terminal o, mejor dicho, *el conjunto de terminales que aparecerían al principio de las cadenas generadas por el no terminal*

Predicción de la regla a aplicar (3)

IMPORTANTE: el conjunto de símbolos que aparecen al principio de las cadenas generadas por un no terminal se conoce con el nombre de *conjunto de PRIMEROS* (*FIRST*), y más adelante veremos cómo calcularlo formalmente.

$$
\begin{array}{lcl}
A & \longrightarrow & B\ C \\
A & \longrightarrow & \textbf{bad} \\
B & \longrightarrow & \textbf{big}\ C\ \textbf{boss} \\
B & \longrightarrow & \textbf{bet} \\
C & \longrightarrow & \textbf{cat} \\
C & \longrightarrow & \textbf{cow}
\end{array}
$$

$$
\begin{array}{lcl}
\text{PRIMEROS}(A) & = & \{\ \textbf{bad}\ ,\ \textbf{big}\ ,\textbf{bet}\ \} \\
\text{PRIMEROS}(B) & = & \{\ \textbf{big}\ ,\textbf{bet}\ \} \\
\text{PRIMEROS}(C) & = & \{\ \textbf{cat}\ ,\textbf{cow}\ \}
\end{array}
$$

Predicción de la regla a aplicar (4)

¿Y si un no terminal genera la cadena vacía ϵ ?

$$
\begin{array}{lcl}
A & \longrightarrow & B\ C \\
A & \longrightarrow & \textbf{bad} \\
B & \longrightarrow & \textbf{big}\ C\ \textbf{boss} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{cat} \\
C & \longrightarrow & \textbf{cow}
\end{array}
$$

$$
\begin{array}{lcl}
\text{PRIMEROS}(A) & = & \{\ \textbf{bad}\ ,\ \textbf{big}\ ,\ \textbf{cat}\ ,\ \textbf{cow}\ \} \\
\text{PRIMEROS}(B) & = & \{\ \textbf{big}\ , \epsilon\} \\
\text{PRIMEROS}(C) & = & \{\ \textbf{cat}\ , \textbf{cow}\ \}
\end{array}
$$

Predicción de la regla a aplicar (5)

Resumen:

- En un momento dado del análisis, se tiene que derivar un no terminal A
- Además, se conoce el símbolo que aparece en la entrada
- Dado el símbolo de la entrada y el no terminal A, se debe elegir qué regla de A hay que aplicar para llegar a un análisis correcto sin *backtracking*
- Para elegir la regla a aplicar, hay que consultar las partes derechas de las reglas de A:
  - Si la parte derecha de una regla empieza por un terminal y ese terminal coincide con el símbolo de la entrada, ésa es la regla que hay que aplicar
  - Si la parte derecha de una regla empieza por un no terminal B, hay que consultar los símbolos que pueden ser generados por B; si el símbolo de la entrada está entre esos símbolos, ésa es la regla que hay que aplicar ¿Y si B genera ϵ? Hay que mirar los símbolos que aparecen después de B en la regla de A

Predicción de la regla a aplicar (6)

¿Y si ...

- ... todos los símbolos de la parte derecha de la regla de A son no terminales y todos generan ϵ ?

$$
\begin{array}{lcl}
A & \longrightarrow & B\ C \\
A & \longrightarrow & \ldots \\
B & \longrightarrow & \epsilon \\
B & \longrightarrow & \ldots \\
C & \longrightarrow & \epsilon \\
C & \longrightarrow & \ldots
\end{array}
$$

- ... o bien la parte derecha de A es directamente ϵ ?

En esos casos, parece que mirando solamente los PRIMEROS no es suficiente para decidir qué regla aplicar

Predicción de la regla a aplicar (7)

Hay dos soluciones al problema de las reglas que generan ϵ :

1. Cuando no se puede aplicar ninguna otra regla, se aplica la regla que genera ϵ (algunos compiladores lo hacen, porque si hay un error en la entrada se detectará más adelante, al emparejar terminales)
2. Mirar, antes de decidir si aplicar la regla que genera ϵ , los símbolos que pueden aparecer después de *A* en una derivación válida
   Ejemplo:

$$
\begin{array}{lcl}
A & \longrightarrow & B\ C \\
A & \longrightarrow & \textbf{ant}\ A\ \textbf{all} \\
B & \longrightarrow & \textbf{big}\ C\ \textbf{bad} \\
B & \longrightarrow & \textbf{bus}\ A\ \textbf{boss} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{cat} \\
C & \longrightarrow & \epsilon
\end{array}
$$

   El conjunto de símbolos que pueden aparecer después de un no terminal *A* en una derivación válida se denomina el *conjunto de SIGUIENTES de A*: SIGUIENTES(A) = {**all** , **boss** , **$** }

Cálculo formal de la regla a predecir: conjunto de predicción

Dado una gramática *G* y un no terminal de la gramática *A*, para poder predecir en cualquier derivación qué regla de *A* se debe aplicar es necesario calcular *el conjunto de predicción de cada regla de A*, PRED($A \longrightarrow \alpha$):

$$
\begin{array}{l}
\text{PRED}(A \longrightarrow \alpha) = \\
\quad \text{si } \epsilon \in \text{PRIMEROS}(\alpha) \text{ entonces} \\
\quad\quad (\text{PRIMEROS}(\alpha) - \{\epsilon\}) \cup \text{SIGUIENTES}(A) \\
\quad \text{si no} \\
\quad\quad \text{PRIMEROS}(\alpha)
\end{array}
$$

Dado un no terminal *A* y un terminal de la entrada *t*, se aplicará la regla de *A* que tenga a *t* en su conjunto de predicción ¿Y si hay más de una regla de *A* que tenga a *t*?

Cálculo formal de la regla a predecir: PRIMEROS

El cálculo del conjunto de PRIMEROS se define formalmente de la siguiente manera:

· PRIMEROS(a) $= \{\ a\ \}$, si a es un terminal

· PRIMEROS(A) $= \cup_{A \to \alpha_i}$PRIMEROS(α_i), si A es un no terminal

· Para calcular PRIMEROS de una cadena α de terminales y no terminales:

- si α es ϵ , entonces PRIMEROS(α) $= \{\epsilon\ \}$
- si $\alpha = a_1 a_2 \ldots a_n$ donde cada a_i puede ser un terminal o un no terminal de la gramática:
  - Si a_1 es un terminal, PRIMEROS(α) $= \{\ a_1\ \}$
  - Si a_1 es un no terminal, hay que añadir PRIMEROS(a_1) $- \{\epsilon\ \}$ a PRIMEROS(α), y si $\epsilon \in$ PRIMEROS(a_1):
    - Si $n = 1$, es decir, $\alpha = a_1$, entonces hay que añadir ϵ a PRIMEROS(α)
    - Si $n > 1$ entonces hay que añadir PRIMEROS($a_2 \ldots a_n$) a PRIMEROS(α)

Cálculo formal de la regla a predecir: SIGUIENTES

El cálculo del conjunto de SIGUIENTES de un no terminal *A* se define de la siguiente manera:

1. Si *A* es el símbolo inicial de la gramática, añadir $ a SIGUIENTES(*A*)
2. Sea una regla de la gramática que contiene en su parte derecha el no terminal *A*, $B \longrightarrow \alpha A \beta$, donde *B* es otro no terminal (que podría ser también *A*), y α y β son cadenas de terminales y no terminales de la gramática (ambas pueden ser ϵ):
   1. Añadir PRIMEROS(β) $- \{\epsilon\}$ a SIGUIENTES(*A*)
   2. Si $\epsilon \in$ PRIMEROS(β) (o bien $\beta = \epsilon$), entonces hay que añadir los SIGUIENTES(*B*) a los SIGUIENTES(*A*)

Ejemplo del cálculo de conjuntos de predicción

$$
\begin{array}{lcl}
A & \longrightarrow & B\ C \\
A & \longrightarrow & \textbf{ant}\ A\ \textbf{all} \\
B & \longrightarrow & \textbf{big}\ C \\
B & \longrightarrow & \textbf{bus}\ A\ \textbf{boss} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{cat} \\
C & \longrightarrow & \textbf{cow}
\end{array}
$$

Para poder calcular el conjunto de PRIMEROS de las partes derechas de las reglas, primero se calculan los PRIMEROS de los no terminales:

$$
\begin{array}{lcl}
\text{PRIMEROS}(A) & = & \{\textbf{ant}\ , \textbf{big}\ , \textbf{bus}\ , \textbf{cat}\ , \textbf{cow}\ \} \\
\text{PRIMEROS}(B) & = & \{\textbf{big}\ , \textbf{bus}\ , \epsilon\ \} \\
\text{PRIMEROS}(C) & = & \{\textbf{cat}\ , \textbf{cow}\ \}
\end{array}
$$

Ejemplo del cálculo de conjuntos de predicción (2)

$$
\begin{array}{lcl}
A & \longrightarrow & B\ C \\
A & \longrightarrow & \textbf{ant}\ A\ \textbf{all} \\
B & \longrightarrow & \textbf{big}\ C \\
B & \longrightarrow & \textbf{bus}\ A\ \textbf{boss} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{cat} \\
C & \longrightarrow & \textbf{cow}
\end{array}
$$

Como hay una producción que genera ϵ , tenemos que calcular los SIGUIENTES de cada no terminal:

$$
\begin{array}{lcl}
\text{SIGUIENTES}(A) & = & \{\textbf{all}\ , \textbf{boss}\ , \textbf{\$}\ \} \\
\text{SIGUIENTES}(B) & = & \{\textbf{cat}\ , \textbf{cow}\ \} \\
\text{SIGUIENTES}(C) & = & \{\textbf{cat}\ , \textbf{cow}\ , \textbf{all}\ , \textbf{boss}\ , \textbf{\$}\ \}
\end{array}
$$

Ejemplo del cálculo de conjuntos de predicción (3)

Los conjuntos de predicción de cada regla son:

A	$\longrightarrow$	$B\ C$	{**big** , **bus** , **cat** , **cow** }	PRIM(BC)
A	$\longrightarrow$	**ant** A **all**	{**ant** }	
B	$\longrightarrow$	**big** C	{**big** }	
B	$\longrightarrow$	**bus** A **boss**	{**bus** }	
B	$\longrightarrow$	ϵ	{**cat** , **cow** }	(PRIM(ϵ) $-$ {ϵ }) $\cup$ SIG(B)
C	$\longrightarrow$	**cat**	{**cat** }	
C	$\longrightarrow$	**cow**	{**cow** }	

Derivación	Entrada	Regla/acción
A **$**	**ant cat all $**	$A \longrightarrow$ **ant** A **all**
ant A **all $**	**ant cat all $**	emparejar **ant**
A **all $**	**cat all $**	$A \longrightarrow B\ C$
$B\ C$ **all $**	**cat all $**	$B \longrightarrow \epsilon$
C **all $**	**cat all $**	$C \longrightarrow$ **cat**
cat all $	**cat all $**	emparejar **cat**
all $	**all $**	emparejar **all**
$	**$**	OK!!

Ejemplo: cálculo de PRIMEROS

$$
\begin{array}{lcl}
S & \longrightarrow & A\ B\ C \\
A & \longrightarrow & D\ E\ \mathbf{f}\ C \mid \mathbf{a} \mid \epsilon \\
B & \longrightarrow & E\ G\ C \mid \mathbf{b} \mid \epsilon \\
C & \longrightarrow & \mathbf{c} \mid \epsilon \\
D & \longrightarrow & \mathbf{d} \mid \epsilon \\
E & \longrightarrow & \mathbf{e} \mid \epsilon \\
G & \longrightarrow & \mathbf{g}
\end{array}
$$

$$
\begin{array}{lcl}
\text{PRIM}(G) & = & \{\mathbf{g}\ \} \\
\text{PRIM}(E) & = & \{\mathbf{e}\ , \epsilon\ \} \\
\text{PRIM}(D) & = & \{\mathbf{d}\ , \epsilon\ \} \\
\text{PRIM}(C) & = & \{\mathbf{c}\ , \epsilon\ \} \\
\text{PRIM}(B) & = & \{\mathbf{b}\ , \epsilon\ , \mathbf{e}\ , \mathbf{g}\ \} \\
\text{PRIM}(A) & = & \{\mathbf{a}\ , \epsilon\ , \mathbf{d}\ , \mathbf{e}\ , \mathbf{f}\ \} \\
\text{PRIM}(S) & = & \{\mathbf{a}\ , \mathbf{d}\ , \mathbf{e}\ , \mathbf{f}\ , \mathbf{b}\ , \mathbf{g}\ , \mathbf{c}\ , \epsilon\ \}
\end{array}
$$

Ejemplo: cálculo de SIGUIENTES

$$
\begin{array}{lcl}
S & \longrightarrow & A\ B\ C \\
A & \longrightarrow & D\ E\ \mathbf{f}\ C \mid \mathbf{a} \mid \epsilon \\
B & \longrightarrow & E\ G\ C \mid \mathbf{b} \mid \epsilon \\
C & \longrightarrow & \mathbf{c} \mid \epsilon \\
D & \longrightarrow & \mathbf{d} \mid \epsilon \\
E & \longrightarrow & \mathbf{e} \mid \epsilon \\
G & \longrightarrow & \mathbf{g}
\end{array}
$$

$$
\begin{array}{lcl}
\mathrm{SIG}(S) & = & \{\$\} \\
\mathrm{SIG}(A) & = & \{\mathbf{b}\ ,\mathbf{e}\ ,\mathbf{g}\ ,\mathbf{c}\ ,\$\} \\
\mathrm{SIG}(B) & = & \{\mathbf{c}\ ,\$\} \\
\mathrm{SIG}(C) & = & \{\$\ ,\mathbf{b}\ ,\mathbf{e}\ ,\mathbf{g}\ ,\mathbf{c}\ \} \\
\mathrm{SIG}(D) & = & \{\mathbf{e}\ ,\mathbf{f}\ \} \\
\mathrm{SIG}(E) & = & \{\mathbf{f}\ ,\mathbf{g}\ \} \\
\mathrm{SIG}(G) & = & \{\mathbf{c}\ ,\$\ \}
\end{array}
$$

Ejemplo: cálculo de conjuntos de predicción

S	$\longrightarrow$	$A\ B\ C$	$\{\mathbf{a}\ ,\mathbf{d}\ ,\mathbf{e}\ ,\mathbf{f}\ ,\mathbf{b}\ ,\mathbf{g}\ ,\mathbf{c}\ ,\$\}$
A	$\longrightarrow$	$D\ E\ \mathbf{f}\ C$	$\{\mathbf{d}\ ,\mathbf{e}\ ,\mathbf{f}\ \}$
A	$\longrightarrow$	$\mathbf{a}$	$\{\mathbf{a}\ \}$
A	$\longrightarrow$	ϵ	$\{\mathbf{b}\ ,\mathbf{e}\ ,\mathbf{g}\ ,\mathbf{c}\ ,\$\}$
B	$\longrightarrow$	$E\ G\ C$	$\{\mathbf{e}\ ,\mathbf{g}\ \}$
B	$\longrightarrow$	$\mathbf{b}$	$\{\mathbf{b}\ \}$
B	$\longrightarrow$	ϵ	$\{\mathbf{c}\ ,\$\}$
C	$\longrightarrow$	$\mathbf{c}$	$\{\mathbf{c}\ \}$
C	$\longrightarrow$	ϵ	$\{\$\ ,\mathbf{b}\ ,\mathbf{e}\ ,\mathbf{g}\ ,\mathbf{c}\ \}$
D	$\longrightarrow$	$\mathbf{d}$	$\{\mathbf{d}\ \}$
D	$\longrightarrow$	ϵ	$\{\mathbf{e}\ ,\mathbf{f}\ \}$
E	$\longrightarrow$	$\mathbf{e}$	$\{\mathbf{e}\ \}$
E	$\longrightarrow$	ϵ	$\{\mathbf{f}\ ,\mathbf{g}\ \}$
G	$\longrightarrow$	$\mathbf{g}$	$\{\mathbf{g}\ \}$

Ejercicio 1

Calcula los conjuntos de PRIMEROS y SIGUIENTES de los no terminales de la siguiente gramática, y los conjuntos de predicción de las reglas:

$$
\begin{array}{lcl}
S & \longrightarrow & A \ \textbf{uno} \ B \ C \\
S & \longrightarrow & S \ \textbf{dos} \\
A & \longrightarrow & B \ C \ D \\
A & \longrightarrow & A \ \textbf{tres} \\
A & \longrightarrow & \epsilon \\
B & \longrightarrow & D \ \textbf{cuatro} \ C \ \textbf{tres} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{cinco} \ D \ B \\
C & \longrightarrow & \epsilon \\
D & \longrightarrow & \textbf{seis} \\
D & \longrightarrow & \epsilon
\end{array}
$$

Ejercicio 2

Calcula los conjuntos de PRIMEROS y SIGUIENTES de los no terminales de la siguiente gramática, y los conjuntos de predicción de las reglas:

$$
\begin{array}{lcl}
S & \longrightarrow & A\ \ B\ \ \textbf{uno} \\
A & \longrightarrow & \textbf{dos}\ \ B \\
A & \longrightarrow & \epsilon \\
B & \longrightarrow & C\ \ D \\
B & \longrightarrow & \textbf{tres} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{cuatro}\ \ A\ \ B \\
C & \longrightarrow & \textbf{cinco} \\
D & \longrightarrow & \textbf{seis} \\
D & \longrightarrow & \epsilon
\end{array}
$$

Condición LL(1)

¿Qué ocurriría si ...

$$\begin{array}{lcll} A & \longrightarrow & \alpha_1 & \{\ldots, \mathbf{a}, \ldots\} \\ A & \longrightarrow & \alpha_2 & \{\ldots, \mathbf{a}, \ldots\} \end{array}$$

... aparece el mismo símbolo en dos o más conjuntos de predicción del mismo no terminal? Evidentemente, cuando en la entrada aparezca el símbolo **a** y el analizador tuviera que derivar el no terminal *A*, no sabría elegir qué regla aplicar.

Condición LL(1): *una gramática G se dice cumple la condición LL(1), si para todos los no terminales, no existen símbolos comunes en los conjuntos de predicción de sus reglas.*

Dicho de otro modo: si existe un símbolo común en los conjuntos de dos reglas del mismo no terminal, se puede decir que la gramática no es LL(1)

IMPORTANTE: para poder hacer un análisis sintáctico descendente en tiempo lineal es necesario que la gramática sea LL(1)

Características no LL(1)

Realmente, para asegurar que una gramática no es LL(1) se debe comprobar que no cumpla la condición LL(1). Sin embargo, hay algunas características que hacen que una gramática no sea LL(1):

1. Recursividad por la izquierda

$$\begin{array}{lcl} A & \longrightarrow & A \ \ldots \\ A & \longrightarrow & \ldots \end{array}$$

2. Factores comunes por la izquierda

$$\begin{array}{lcl} A & \longrightarrow & \alpha \ \beta_1 \\ \ldots & & \\ A & \longrightarrow & \alpha \ \beta_m \end{array}$$

3. Ambigüedad

Condición LL(1) y ambigüedad

MUY IMPORTANTE:

- si una gramática no es LL(1), NO SIEMPRE va a ser ambigua.
- La única forma de determinar que una gramática es ambigua es encontrando dos o más árboles de derivación para una cadena.
- Si una gramática es LL(1), podemos afirmar que no es ambigua. Si no es LL(1), puede que sea ambigua, o puede que no lo sea

$$
\begin{array}{lcl}
A & \longrightarrow & B \ \textbf{uno} \\
A & \longrightarrow & \textbf{dos} \\
B & \longrightarrow & \textbf{dos}
\end{array}
$$

No es LL(1), y no es ambigua

Condiciones LL(k) y LL(*)

- La condición LL(1) es la condición que deben cumplir las gramáticas para las que se quiera hacer un análisis lineal de izquierda a derecha (Left-to-right), usando la derivación por la izquierda (Leftmost derivation) y mirando solamente 1 símbolo de la entrada.
- Si en lugar de mirar un símbolo de la entrada se miraran k símbolos ($k > 1$), sería posible realizar el análisis en tiempo lineal para un conjunto más amplio de gramáticas. Por ejemplo:

$$\begin{array}{lcl} A & \longrightarrow & B \ \textbf{uno} \\ A & \longrightarrow & \textbf{dos} \\ B & \longrightarrow & \textbf{dos} \end{array}$$

  No es LL(1), pero sí es LL(2).
- Existe una condición, LL(*), que es más general que la condición LL(k), y que permite realizar el análisis lineal con un conjunto todavía mayor de gramáticas

Transformaciones para conseguir la condición LL(1)

La recursividad por la izquierda y los factores comunes por la izquierda se pueden *eliminar* de una gramática sin mucha dificultad. Sin embargo, la ambigüedad (si se ha detectado) no es fácil de eliminar y requiere un rediseño manual de la gramática. Además, no siempre que la gramática no es LL(1) es porque es ambigua.

- Eliminación de la recursividad por la izquierda:

$$
\begin{array}{lcl}
A & \longrightarrow & A\ \alpha_1 \\
\ldots & & \\
A & \longrightarrow & A\ \alpha_n \\
A & \longrightarrow & \beta_1 \\
\ldots & & \\
A & \longrightarrow & \beta_m
\end{array}
\qquad
\begin{array}{lcl}
A & \longrightarrow & \beta_1 A' \\
\ldots & & \\
A & \longrightarrow & \beta_m A' \\
A' & \longrightarrow & \alpha_1 A' \\
\ldots & & \\
A' & \longrightarrow & \alpha_n A' \\
A' & \longrightarrow & \epsilon
\end{array}
$$

Transformaciones para conseguir la condición LL(1) (2)

- Ejemplo de eliminación de la recursividad por la izquierda:

$$
\begin{array}{lcl}
E & \longrightarrow & E \ \textbf{opsuma} \ T \\
E & \longrightarrow & T
\end{array}
$$

$$
\begin{array}{lcl}
\alpha_1 & \equiv & \textbf{opsuma} \ T \\
\beta_1 & \equiv & T
\end{array}
$$

$$
\begin{array}{lcl}
E & \longrightarrow & T \ E' \\
E' & \longrightarrow & \textbf{opsuma} \ T \ E' \\
E' & \longrightarrow & \epsilon
\end{array}
$$

Transformaciones para conseguir la condición LL(1) (3)

- Eliminación de los factores comunes por la izquierda:

$$\begin{array}{lcl} A & \longrightarrow & \alpha\beta_1 \\ \ldots & & \\ A & \longrightarrow & \alpha\beta_m \end{array} \qquad \begin{array}{lcl} A & \longrightarrow & \alpha A' \\ A' & \longrightarrow & \beta_1 \\ \ldots & & \\ A' & \longrightarrow & \beta_m \end{array}$$

- Ejemplo:

$$\begin{array}{lcl} \textit{Inst} & \longrightarrow & \textbf{if } E \textbf{ then } \textit{Inst} \textbf{ endif} \\ \textit{Inst} & \longrightarrow & \textbf{if } E \textbf{ then } \textit{Inst} \textbf{ else } \textit{Inst} \textbf{ endif} \end{array}$$

$$\begin{array}{lcl} \textit{Inst} & \longrightarrow & \textbf{if } E \textbf{ then } \textit{Inst } \textit{Inst'} \\ \textit{Inst'} & \longrightarrow & \textbf{endif} \\ \textit{Inst'} & \longrightarrow & \textbf{else } \textit{Inst} \textbf{ endif} \end{array}$$

Analizador sintáctico descendente recursivo

- Una forma muy común de realizar un análisis lineal con una gramática LL(1) es utilizando un analizador sintático descendente recursivo (ASDR)
- Antes de programar el ASDR, es necesario calcular los conjuntos de predicción de todas las reglas y comprobar que la gramática es LL(1)
- En un ASDR, se debe diseñar una función para cada no terminal de la gramática
- Se utiliza una función auxiliar para emparejar terminales
- Cuando se tiene que derivar un no terminal, se llama a la función asociada a ese no terminal, y es la función la que se encarga de analizar el sublenguaje generado por dicho no terminal

Analizador sintáctico descendente recursivo (2)

Ejemplo:

$$
\begin{array}{lcl}
A & \longrightarrow & B \ \textbf{uno} \\
A & \longrightarrow & \textbf{dos} \\
B & \longrightarrow & \textbf{tres} \\
B & \longrightarrow & \textbf{cuatro} \ A
\end{array}
$$

```
public final void A()
{
  if (token.tipo == Token.TRES || token.tipo == Token.CUATRO) {
     B();
     emparejar(Token.UNO);
  }
  else if (token.tipo == Token.DOS) {
     emparejar(Token.DOS);
  }
  else errorSintaxis(Token.TRES,Token.CUATRO,Token.DOS);
}
```

Analizador sintáctico descendente recursivo (3)

```
class AnalizadorSintactico {
...
 public final void emparejar(int tokEsperado)
 {
   if (token.tipo == tokEsperado)
      token = lexico.getNextToken();
   else
      errorSintaxis(tokEsperado);
 }
}
class Compilador {
public void Main(...) {
   ...
   AnalizadorLexico al = new AnalizadorLexico(ficheroentrada)
   AnalizadorSintactico asdr = new AnalizadorSintactico(al);

   asdr.A();  // símbolo inicial de la gramática
   asdr.comprobarFinFichero();
    //  if (token.tipo != Token.FINFICHERO)
    //     errorSintaxis(Token.FINFICHERO);
}
```

Ejercicio 3

Dada la siguiente gramática:

$$
\begin{array}{lcl}
S & \longrightarrow & A\ B\ C \\
S & \longrightarrow & D\ E \\
A & \longrightarrow & \textbf{dos}\ B\ \textbf{tres} \\
A & \longrightarrow & \epsilon \\
B & \longrightarrow & B\ \textbf{cuatro}\ C\ \textbf{cinco} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{seis}\ A\ B \\
C & \longrightarrow & \epsilon \\
D & \longrightarrow & \textbf{uno}\ A\ E \\
D & \longrightarrow & B \\
E & \longrightarrow & \textbf{tres}
\end{array}
$$

1. Calcula los conjuntos de PRIMEROS de cada no terminal.
2. Calcula los conjuntos de SIGUIENTES de cada no terminal.
3. Calcula los conjuntos de predicción de cada regla.
4. Di si la gramática es LL(1) o no y por qué.
5. Escribe la función del no terminal A en un ASDR.

Ejercicio 4

Dada la siguiente gramática:

$$
\begin{array}{lcl}
S & \longrightarrow & B \ \textbf{uno} \\
S & \longrightarrow & \textbf{dos} \ C \\
S & \longrightarrow & \epsilon \\
A & \longrightarrow & S \ \textbf{tres} \ B \ C \\
A & \longrightarrow & \textbf{cuatro} \\
A & \longrightarrow & \epsilon \\
B & \longrightarrow & A \ \textbf{cinco} \ C \ \textbf{seis} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{siete} \ B \\
C & \longrightarrow & \epsilon
\end{array}
$$

1. Calcula los conjuntos de PRIMEROS de cada no terminal.
2. Calcula los conjuntos de SIGUIENTES de cada no terminal.
3. Calcula los conjuntos de predicción de cada regla.
4. Di si la gramática es LL(1) o no y por qué.
5. Escribe la función del no terminal B en un ASDR.

Ejercicio 5

Dada la siguiente gramática:

$$
\begin{array}{lcl}
S & \longrightarrow & A\ B\ C \\
S & \longrightarrow & S\ \textbf{uno} \\
A & \longrightarrow & \textbf{dos}\ B\ C \\
A & \longrightarrow & \epsilon \\
B & \longrightarrow & C\ \textbf{tres} \\
B & \longrightarrow & \epsilon \\
C & \longrightarrow & \textbf{cuatro}\ B \\
C & \longrightarrow & \epsilon
\end{array}
$$

1. Calcula los conjuntos de PRIMEROS de cada no terminal.
2. Calcula los conjuntos de SIGUIENTES de cada no terminal.
3. Calcula los conjuntos de predicción de cada regla.
4. Di si la gramática es LL(1) o no y por qué.
5. Escribe la función del no terminal C en un ASDR.