# Las Gramáticas LL

## Gramáticas con Parsing Eficiente

Universidad de Cantabria

# Outline

1. Las Gramáticas LL(k)
2. Extensión del Cálculo de FIRST
3. Tablas de Análisis Sintáctico LL(1)

# Formalización del Concepto LL

## Definición

*Una gramática libre de contexto* $G = (V, \Sigma, Q_0, P)$ *se dice de clase* $LL(k)$ *si verifica la siguiente propiedad: Dadas dos derivaciones, donde* $\omega \in \Sigma^*, A \in V, \alpha, \beta, \gamma \in (V \cup \Sigma)^*$*, del tipo siguiente:*

- $Q_0 \vdash_{lm} \omega A \gamma \vdash_{lm} \omega \alpha \gamma \vdash \omega x \in \Sigma^*$,
- $Q_0 \vdash_{lm} \omega A \gamma \vdash_{lm} \omega \beta \gamma \vdash \omega y \in \Sigma^*$,

*Si* $FIRST_k(x) = FIRST_k(y)$*, entonces* $\alpha = \beta$.

# Idea

La idea es que si hacemos dos derivaciones a izquierda desde una variable de nuestra gramática, y si llegamos a dos formas terminales en las que los primeros $k$ símbolos a partir de $A$ de una forma terminal coinciden, entonces es que hemos tenido que hacer la misma derivación desde $A$.

# Observación

Propiedades:

- Son gramáticas no ambiguas.
- Existe una tabla que permite generar árboles sintácticos para cualquier palabra con un número de operaciones proporcional a la longitud.

# Ejemplos

## Ejemplo

*Un ejemplo de gramática LL(1) es la dada mediante:*
$Q_0 \mapsto aAQ_0 \mid b, A \mapsto a \mid bQ_0A$

# Ejemplos

## Ejemplo

*La gramática* $\{Q_0 \mapsto \lambda \mid abA, A \mapsto Q_0aa \mid b\}$ *es una gramática* $LL(2)$

# Ejemplos

Hay gramáticas que no son $LL(k)$ para ningún $k$.

$$\{Q_0 \mapsto Acd \mid Ac, A \mapsto aA|b\}.$$

# Teorema

## Teorema

*Una gramática* $G = (V, \Sigma, Q_0, P)$ *es* $LL(k)$ *si y solamente si se verifica la siguiente propiedad:*
*Dadas dos producciones* $A \mapsto \beta$ *y* $A \mapsto \gamma$ *tales que* $A$ *es accesible y se tiene* $Q_0 \vdash_{lm} \omega A \alpha$, *con* $\omega \in \Sigma^*$ *y* $\alpha \in (V \cup \Sigma)^*$, *entonces*

$$FIRST_k(\beta\alpha) \cap FIRST_k(\gamma\alpha) = \emptyset.$$

# Idea

Como nos dicta la intuición, en las gramáticas $LL(k)$ tendremos que calcular $FIRST_k(\alpha)$, donde $\alpha$ es una forma no terminal.

# Propiedades

## Definición

*Sea $L_1, L_2 \in \Sigma^*$, dos lenguajes definimos:*

$$L_1 \oplus_k L_2 = \left\{ \omega : \exists x \in {}_1,\ \exists y \in {}_2 \left\{ \begin{array}{l} |xy| \leq k \text{ y } \omega = xy, \text{ o} \\ w = FIRST_k(xy). \end{array} \right. \right\}$$

# Resultado

## Teorema

*Dada una gramática libre de contexto G y una forma sentencial* $\alpha\beta$ *se tiene que*

$$FIRST_k(\alpha\beta) = FIRST_k(\alpha) \oplus_k FIRST_k(\beta).$$

# Algoritmo

**Definir** $F_i(a) = a$ para todo símbolo del alfabeto y para todo $0 \leq i \leq k$.
**Definir** $F_0(A) = \{x \in \Sigma^k : A \mapsto x\alpha\}$ para todo símbolo del alfabeto y para todo $0 \leq i \leq k$.
**Para** $1 \leq i \leq k$ **y mientras** $F_{i-1}(A) \neq F_i(A)$ **para alguna variable** $A$ **hacer**

**Para cada variable A hacer**

$$F_i(A) = \{x \in \Sigma^k : A \mapsto Y_1 \dots Y_n \text{ y } x \in F_{i-1}(Y_1) \oplus_k \cdots \oplus_k F_{i-1}(Y_n)\}.$$

**fin hacer**

**fin hacer**

# Antes de Comenzar..

Suponemos enumeradas nuestras producciones, asignándole un número natural a cada una de ellas.
Además, introduciremos un nuevo símbolo § que hará las funciones de fondo de la pila.

# Objetivo

Construiremos una tabla

$$M : (V \cup \Sigma \cup \{\S\}) \times (\Sigma \cup \{\lambda\}) \longrightarrow \mathcal{P}(P),$$

donde $\mathcal{P}(P)$ es el conjunto de todos los subconjuntos del conjunto de las producciones.

# Algoritmo

La tabla será construida a partir de la gramática dada y cuyas filas están indicadas por los elementos de $V \cup \Sigma \cup \{\S\}$ y cuyas columnas están indicadas por los elementos de $\Sigma \cup \{\lambda\}$.

# Algoritmo

- Dada una producción $(i)$ $A \mapsto \alpha$
  - Para cada $a \in FIRST(\alpha),\ a \neq \lambda$, añadir i a la casilla $M(A, a)$.
  - Si $\lambda \in FIRST(\alpha)$ añadir $i$ en todas las casillas $M(A, b)$ para cada $b \in FOLLOW(A)$.
- $M(a, a) =$ **pop** para cada $a \in \Sigma$.
- $M(\S, \lambda) =$ **accept**.
- En todos los demás casos escribir $M(X, i) =$ **error**.

# Resultado

## Teorema

*Dada una gramática libre de contexto G, y dada T(G) la tabla construida por el algoritmo anterior, entonces G es LL(1) si y solamente si todos las casillas de la tabla T(G) contienen exactamente una producción o una de las palabras seleccionadas (***pop, accept, error***).*

# Ejemplo

Como ejemplo, consideremos la gramática $G = (V, \Sigma, Q_0, P)$, donde las producciones son:

$$P := \{Q_0 \mapsto aAQ_0 \mid b, A \mapsto a \mid bQ_0A\}.$$

Enumeramos estas producciones del modo siguiente:

$$\begin{aligned} (1) \quad & Q_0 \mapsto aAQ_0 \\ (2) \quad & Q_0 \mapsto b \\ (3) \quad & A \mapsto a \\ (4) \quad & A \mapsto bQ_0A \end{aligned}$$

# Ejemplo

| | $a$ | $b$ | $\lambda$ |
|---|---|---|---|
| $Q_0$ | 1 | 2 | **error** |
| $A$ | 3 | 4 | **error** |
| $a$ | **pop** | **error** | **error** |
| $b$ | **error** | **pop** | **error** |
| § | **error** | **error** | **accept** |

# Observaciones Finales

- Las gramáticas $LL(1)$ son la forma más natural de realizar el parsing. Algunos lenguajes de programación están definidos por una gramática $LL(1)$.
- Encontrar una gramática que sea $LL(1)$ es un problema difícil, pero hay producciones que no pueden estar en gramáticas $LL(1)$.
- Es factible testar si una gramática es $LL(1)$.
- Todavia queda por ver como recuperar el árbol de derivación.