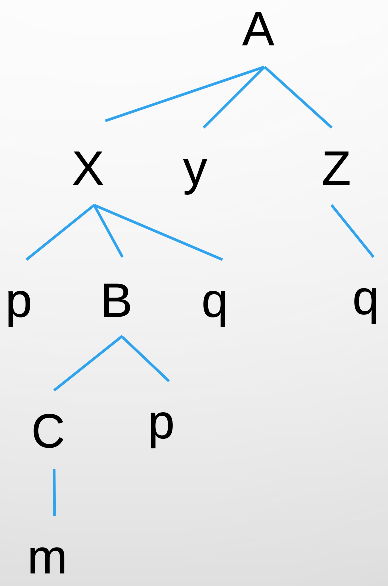
ANÁLISIS SINTÁCTICO DESCENDENTE SIN RETROCESO (GRAMÁTICAS LL)

Descendente

- Construye el árbol desde la raíz (axioma de la GCL) hacia las hojas
- Funciona por derivaciones a la izquierda

1. A → X y Z
2. X → p B q
3. B → C p
4. C → m
5. Z → q



La GCL no puede ser recursiva por la izquierda → bucle infinito!!

parse: 1 2 3 4 5

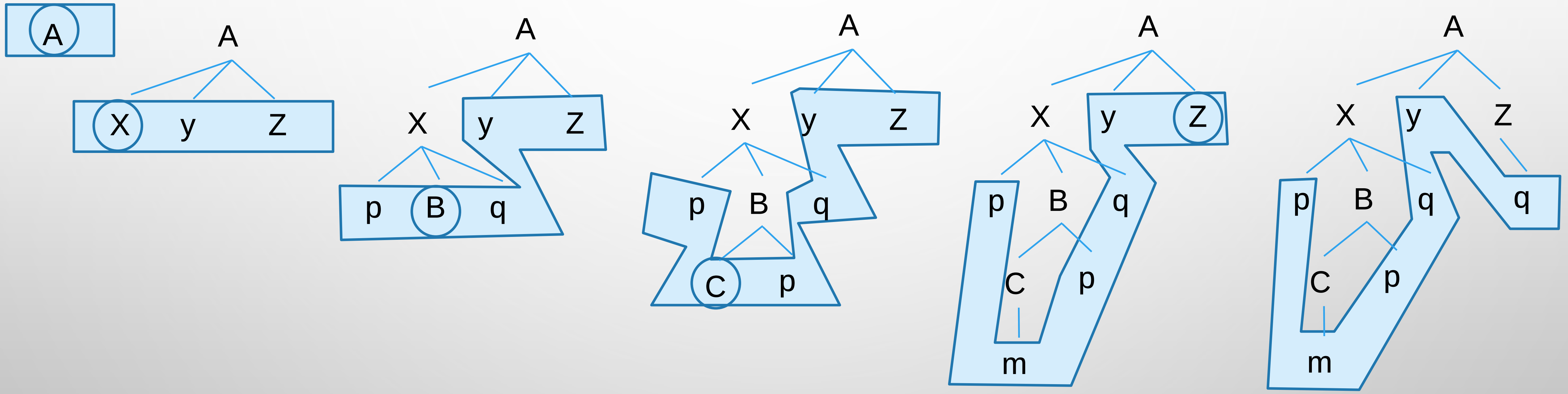
Descendente sin retroceso

- En cada instante solo hay una regla válida. ¿Qué propiedad de la GCL garantiza esto? → **Gramáticas LL**

Derivaciones a la izquierda y Forma Sentencial

1. A → X y Z
2. X → p B q
3. Z → q
4. B → C p
5. C → m

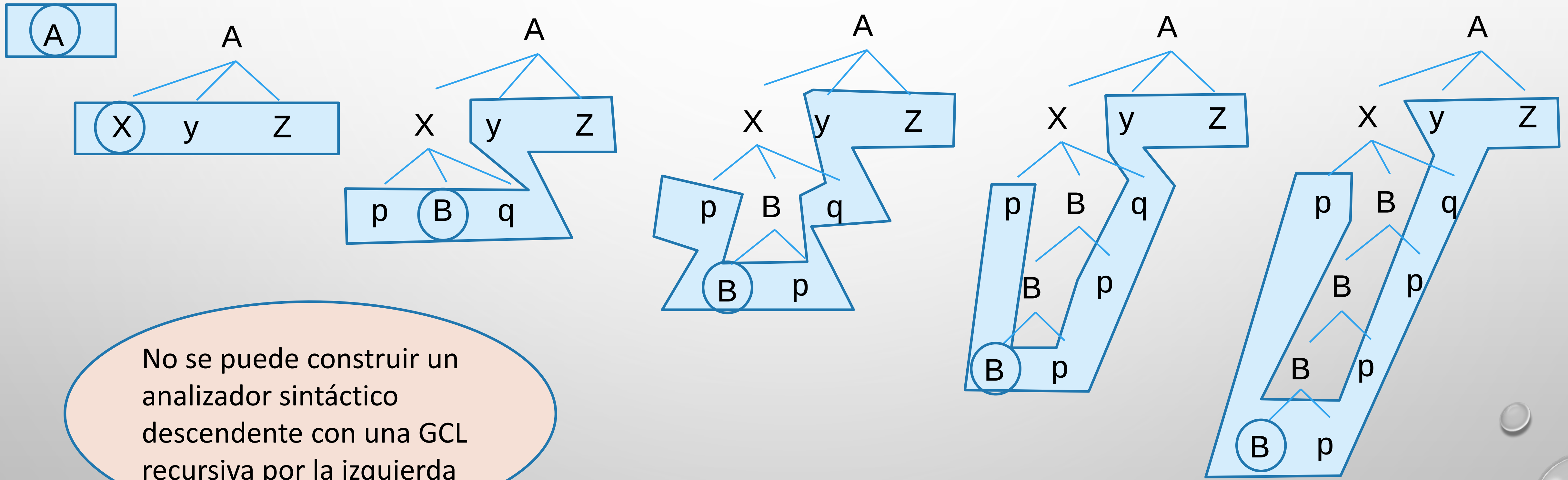
ω = p m p q y q



Problema con las GCL recursivas por la izquierda

1. A → X y Z
2. X → p B q
3. Z → q
4. B → B p
5. B → m

ω = p m p q y q



No se puede construir un analizador sintáctico descendente con una GCL recursiva por la izquierda

¡¡ Bucle infinito !!

¿Cómo **eliminar la recursividad por la izquierda**?

G:

$A \rightarrow A\ \alpha$
$A \rightarrow \beta$

¿Qué lenguaje genera esta gramática?

¿Cómo **eliminar la recursividad por la izquierda**?

G:

$A \rightarrow A\,\alpha$
$A \rightarrow \beta$

$L(G) = \{\beta, \beta\,\alpha\,, \beta\,\alpha\,\alpha\,, \beta\,\alpha\,\alpha\,\alpha\,, \beta\,\alpha\,\alpha\,\alpha\,\alpha, \ldots\} = \{\beta\,\alpha^*\}$

G':

$A \rightarrow \beta\,A'$
$A' \rightarrow \alpha\,A'$
$A' \rightarrow \lambda$

¿Cómo **eliminar la recursividad por la izquierda**?

G:

$A \rightarrow A\ \alpha$
$A \rightarrow \beta$

$L(G) = \{\beta, \beta\ \alpha, \beta\ \alpha\ \alpha, \beta\ \alpha\ \alpha\ \alpha, \beta\ \alpha\ \alpha\ \alpha\ \alpha, \ldots\} = \{\beta\ \alpha^*\}$

G':

$A \rightarrow \beta\ A'$
$A' \rightarrow \alpha\ A'$
$A' \rightarrow \lambda$

- Ejemplo G_1:

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow id$

¿Cómo **eliminar la recursividad por la izquierda**?

G:

$A \rightarrow A\ \alpha$
$A \rightarrow \beta$

$L(G) = \{\beta, \beta\ \alpha, \beta\ \alpha\ \alpha, \beta\ \alpha\ \alpha\ \alpha, \beta\ \alpha\ \alpha\ \alpha\ \alpha, \dots\} = \{\beta\ \alpha^*\}$

G':

$A \rightarrow \beta\ A'$
$A' \rightarrow \alpha\ A'$
$A' \rightarrow \lambda$

- Ejemplo G_1:

$E \rightarrow E\ \boxed{+\ T}$ → α
$E \rightarrow \boxed{T}$ → β
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow id$

$E \rightarrow \boxed{T}\ E'$
$E' \rightarrow \boxed{+\ T}\ E' \mid \lambda$
$T \rightarrow T\ \boxed{*\ F}$ → α
$T \rightarrow \boxed{F}$ → β
$F \rightarrow id$

G'_1:

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \lambda$
$T \rightarrow \boxed{F}\ T'$
$T' \rightarrow \boxed{*\ F}\ T' \mid \lambda$
$F \rightarrow id$

¿Cómo **eliminar la recursividad por la izquierda**?

G:

$A \rightarrow A\ \alpha$
$A \rightarrow \beta$

G':

$A \rightarrow \beta\ A'$
$A' \rightarrow \alpha\ A'$
$A' \rightarrow \lambda$

$L(G) = \{\beta, \beta\ \alpha\ , \beta\ \alpha\ \alpha\ , \beta\ \alpha\ \alpha\ \alpha\ , \ldots\ \} = \{\beta\ \alpha^*\}$

Ejemplo G_1:

$E \rightarrow E\ \boxed{+\ T}$
$E \rightarrow \boxed{T}$
$T \rightarrow T\ *\ F$
$T \rightarrow F$
$F \rightarrow id$

$\alpha = +\ T$
$(E \rightarrow E\ \alpha)$

$\beta = T$
$(E \rightarrow \beta)$

$E \rightarrow \boxed{T}\ E'$
$E' \rightarrow \boxed{+\ T}\ E' \mid \lambda$
$T \rightarrow T\ \boxed{*\ F}$
$T \rightarrow \boxed{F}$
$F \rightarrow id$

$E \rightarrow \beta\ E'$
$E' \rightarrow \alpha\ E'$
$E' \rightarrow \lambda$

$\alpha = *\ F$
$(T \rightarrow T\ \alpha)$

$\beta = F$
$(T \rightarrow \beta)$

G'_1:

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \lambda$
$T \rightarrow \boxed{F}\ T'$
$T' \rightarrow \boxed{*\ F}\ T' \mid \lambda$
$F \rightarrow id$

$T \rightarrow \beta\ T'$
$T' \rightarrow \alpha\ T'$
$T' \rightarrow \lambda$

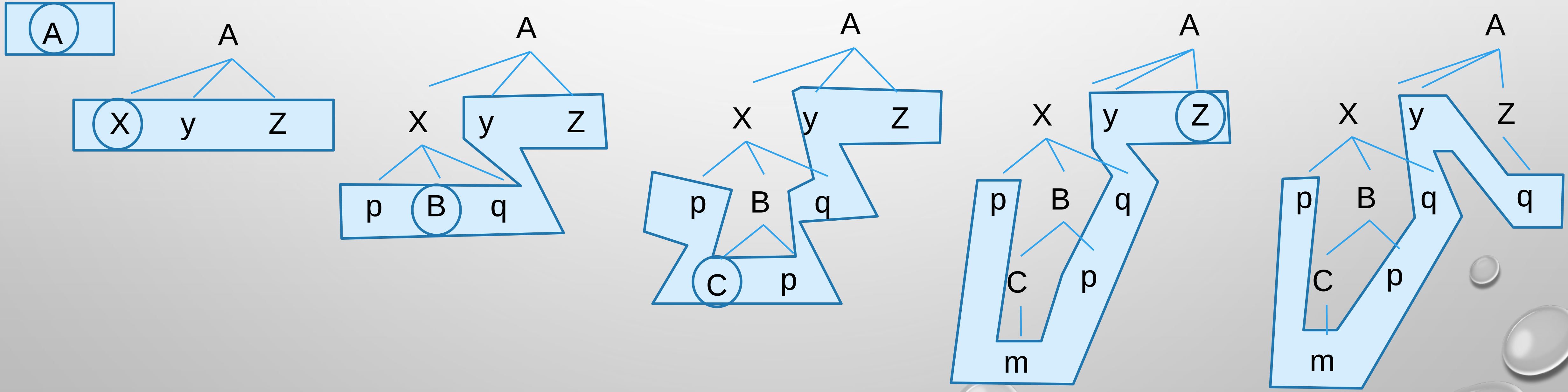
Descendente sin retroceso

Cuando hay que aplicar una derivación, **solo hay una regla posible**

- No terminal a expandir? → derivaciones a la izquierda
- Regla a aplicar? → **siguiente token** (→ gramática LL(1))

1. A → X y Z
2. X → p B q
3. Z → q
4. B → C p
5. C → m

ω = p m p q y q



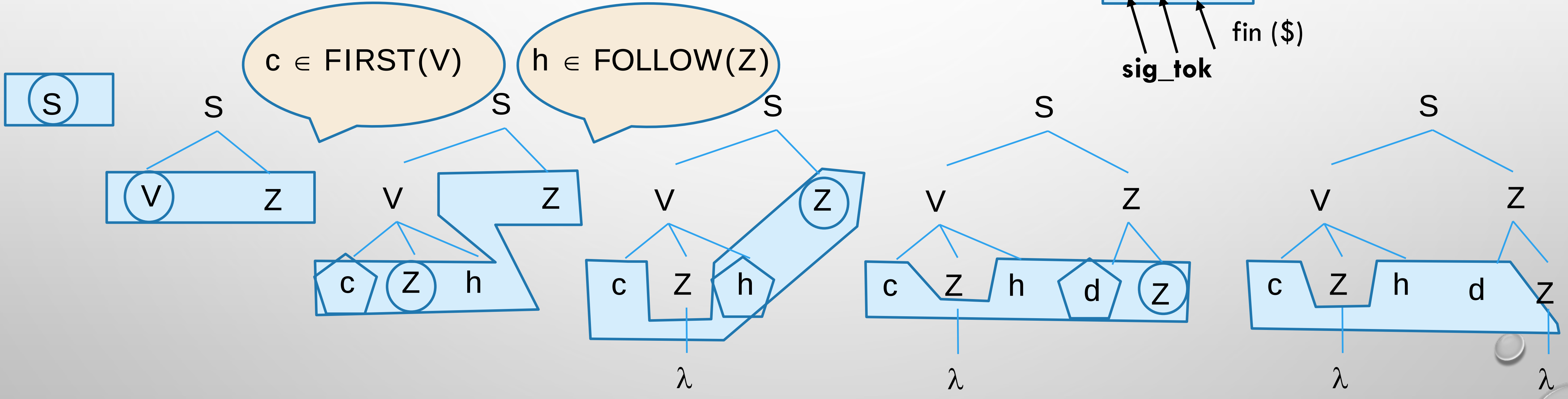
Descendente sin retroceso

$S \rightarrow T\ V \mid V\ Z$
$T \rightarrow a\ T \mid b\ T \mid h$
$V \rightarrow \lambda \mid c\ Z\ h$
$Z \rightarrow \lambda \mid d\ Z$

Cuando hay que aplicar una derivación, **solo hay una regla posible**

- No terminal a expandir? → derivaciones a la izquierda
- Regla a aplicar? → **siguiente token**

$\omega =$ c h d



EL SIGUIENTE TOKEN DETERMINA CUÁL ES LA REGLA A APLICAR → Gramáticas LL(1)

Descendente sin retroceso

Cuando hay que aplicar una derivación, **solo hay una regla posible**

- Gramáticas **LL(k)**. Permiten saber qué regla hay que aplicar conociendo, como máximo, los k siguientes tokens de la entrada
- Gramáticas **LL(1)**. Sólo se necesita conocer **un token**.

Construcción del árbol. Se mira el símbolo del nodo activo:

- Si es un terminal, y coincide con **el token actual**, se equiparan (se pide al A. Léx. el siguiente token y se avanza al siguiente nodo del árbol). Si no hubieran coincidido, se habría detectado un error sintáctico.
- Si es un No terminal, se aplica la única regla de derivación que nos llevará a obtener **el token actual**.
  - ✓ a lo sumo, una regla permitirá obtener, desde ese No terminal, el token actual como "primer símbolo terminal más a la izquierda". → FIRST
  - ✓ Y ¿qué pasaría si para ese No terminal existe una regla lambda? ¿Con quién se equipara el token actual de la cadena de entrada? Con el símbolo terminal que vaya a continuación en la forma sentencial". → FOLLOW

FIRST

FIRST(X), donde ($X \in \{T \cup N\}$), o
FIRST(α), donde ($\alpha \in \{T \cup N\}^*$)

Conjunto formado por los Terminales que pueden aparecer como **primer símbolo terminal** en las cadenas derivadas a partir de *X* (o a partir de α).

FOLLOW

FOLLOW(A), donde ($A \in N$)

Conjunto formado por los Terminales que pueden aparecer **inmediatamente a continuación** de *A* en alguna forma sentencial.