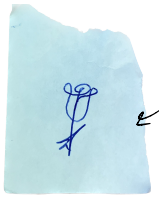


Sintáctico: Se encarga de que los tokens estén en el orden adecuado

REGLAS:
Las expresiones usan notación infija
El número de paréntesis abiertos es siempre igual al de paréntesis cerrados.

Mayúscula → TOKEN → Z
minúscula → no terminal → a

Lenguaje: el lenguaje que genera una gramática es el conjunto de todas sus sentencias.
Conjunto de todos los posibles programas

$$S \rightarrow S + S$$
$$S \rightarrow X$$

$S \rightarrow X$

$S \rightarrow S\ '+'\ S \rightarrow X + X$

$S \rightarrow S\ '+'\ S \rightarrow S + S + S$

genera → $X + \ldots + X$

$$S \rightarrow X\ mt \rightarrow X$$
$$S \rightarrow X\ mt \rightarrow X + X\ mt$$

$$S \rightarrow X\ mt \quad +X$$
$$mt \rightarrow +\ X\ mt$$
$$mt \rightarrow \lambda$$

SON GRAMÁTICAS EQUIVALENTES

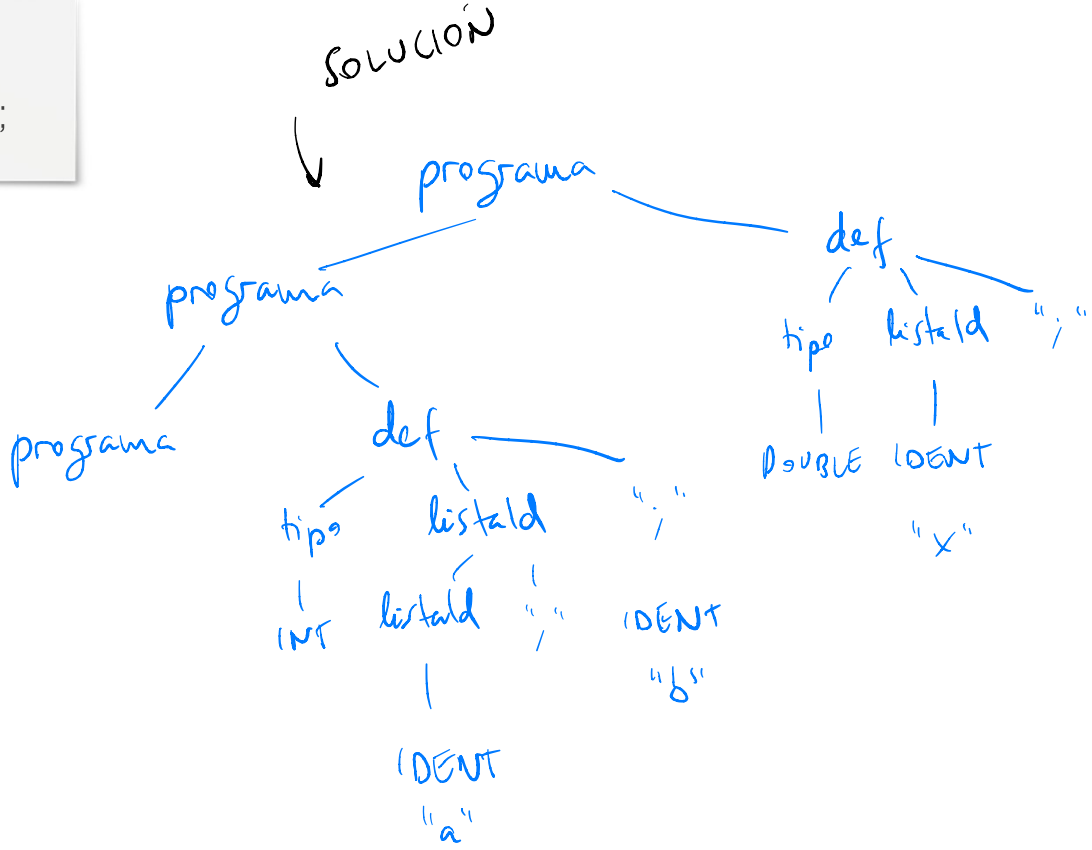
→ Dado cualquier lenguaje, existen infinitas gramáticas equivalentes

Decir si es una sentencia válida:

programa → programa def | λ
def → tipo listaId ';'
tipo → INT | DOUBLE
listaId → IDENT | listaId ',' IDENT

int a, b;
double x;

SOLUCIÓN



programa
↓
def
↓
tipo (INT)
↓
listaId (IDENT, IDENT)
↓
tipo (DOUBLE)
↓
listaId (IDENT)

CICLO DE EJECUCIÓN

TIPOS DE ALGORITMOS

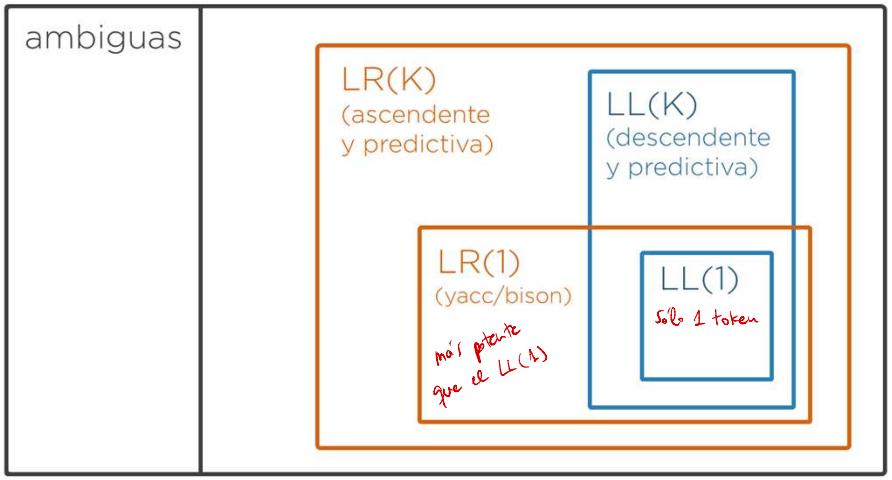
Dirección

- descendente
- ascendente

Selección

- backtracking
- predictivo

TIPOS DE GRAMÁTICAS



prog → decl decl

decl → variables | typedef

variables → VAR ID restolds

restolds → ',' ID restolds
| ':' tipo

tipo → INT | REAL

typedef → TYPE ID restolds

```
• void prog ( ) {
    decl ( );
    decl ( );
  }
• void decl ( ) {
    if ( token.getType == VAR.ID )
      variables ( );
    else
      typedef ( );
  }
• void variables ( ) {
    restolds ( );
  }
• void restolds ( ) {
```

```
• void tipo ( ) {
  if ( token.getType ( ) == INT ) {
      int ( );
  } else
      double ( );
  }
• void typedef ( ) {
    restolds ( )
  }
```

está mal
mirar apuntes