

The background of the slide is a light gray gradient, decorated with numerous realistic water droplets of various sizes. Some droplets are at the top, some at the bottom, and some on the sides, creating a clean, modern aesthetic.

# ANÁLISIS SINTÁCTICO DESCENDENTE PREDICTIVO CON GRAMÁTICAS LL(1)



## Analizador Sintáctico Descendente Predictivo

- ¿Qué gramáticas vamos a usar?

→ LL(1)

- ¿Qué gramáticas no son válidas?

→ recursiva por la izquierda

Para ningún descendente

→ ambigua (salvo alguna excepción aceptada por convenio)

Para ningún sintáctico

→ no factorizada (el consecuente de dos o más reglas de un No terminal comienza igual)

No cumplen la condición LL(1)

- Factorización por la izquierda

→ Para cada No terminal  $A$  que no esté factorizado, encontrar el prefijo común más largo y "sacar factor común"

$G: A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \mid \dots \mid \alpha \beta_n \mid \gamma$

$G': A \rightarrow \alpha A' \mid \gamma$

$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

Si no está factorizada, tenemos que factorizarla



## Ejemplo de factorización de una $G$

$G$ :

$$A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \mid \dots \mid \alpha \beta_n \mid \gamma$$

$G'$ :

$$A \rightarrow \alpha A' \mid \gamma$$
$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$
$$S \rightarrow \text{id} := E + E$$
$$S \rightarrow \text{id} := E * E$$
$$E \rightarrow ( E )$$
$$E \rightarrow \text{id}$$
$$S \rightarrow \text{id} := E S'$$
$$S' \rightarrow + E$$
$$S' \rightarrow * E$$
$$E \rightarrow ( E )$$
$$E \rightarrow \text{id}$$

# ANÁLISIS SINTÁCTICO DESCENDENTE CON GRAMÁTICAS LL(1)

## ANALIZADOR SINTÁCTICO DESCENDENTE PREDICTIVO RECURSIVO

```
...  
7.  $F \rightarrow ( E )$   
8.  $F \rightarrow id$   
...
```

```
Function F ()  
{  
  if sig_tok == '(' then  
  {  
    equipara ();  
    E;  
    equipara ();  
  }  
  else  
  {  
    equipara (id)  
  }  
}
```



## Analizador Sintáctico Descendente Recursivo Predictivo

- Una función para cada símbolo No terminal
  - Cuerpo de la función: if-then-else anidados (una rama para cada regla posible)
  - El token recibido determina qué rama se ejecuta (y se comienza a recorrer el consecuente de la regla). Para cada símbolo del consecuente:
    - Si es un No terminal, se llama a su función
    - Si es un terminal, se equipara (si coincide con el token recibido, se le pide un nuevo token al A. Léxico; si no coinciden, hay un error sintáctico)
- Las funciones se van llamando recursivamente.
- Main del Analizador Sintáctico
  - El A. Sintáctico pide un token al A. Léxico y llama a la función del axioma
  - Al terminar la ejecución de la primera función llamada (la del axioma), si la cadena se ha leído entera el A. Sintáctico termina con éxito; en caso contrario, hay un error sintáctico

```
Function A_Sint ()  
{  
    sig_tok := ALex()  
    E;  
    if sig_tok ≠ '$' then error ()  
}
```



## Ejemplo de construcción de un Recursivo Predictivo

```
Function A_Sint ()  
{  
    sig_tok := ALex()  
    E;  
    if sig_tok ≠ '$' then error ()  
}
```

```
Function equipara (t)  
{  
    if sig_tok == t  
    then sig_tok := ALex()  
    else error ()  
}
```

Una función para cada no terminal (y una rama para cada regla)

No hay if porque solo hay una regla

```
Function E ()  
{  
    print(1);  
    T;  
    E'  
}
```

```
Function T ()  
{  
    print(4);  
    F;  
    T'  
}
```

1.  $E \rightarrow T E'$
2.  $E' \rightarrow + T E'$
3.  $E' \rightarrow \lambda$
4.  $T \rightarrow F T'$
5.  $T' \rightarrow * F T'$
6.  $T' \rightarrow \lambda$
7.  $F \rightarrow ( E )$
8.  $F \rightarrow id$

```
Function E' ()  
{  
    if sig_tok == '+' then  
    { print(2);  
      equipara (+);  
      T;  
      E'  
    }  
    else if sig_tok ∈ {(), $} /* Follow(E')  
    then print(3)  
    else error ()  
}
```

```
Function T' ()  
{  
    if sig_tok == '*' then  
    { print(5);  
      equipara (*);  
      F;  
      T'  
    }  
    else if sig_tok ∈ {+, ), $} /* Follow(T')  
    then print(6)  
    else error ()  
}
```

```
Function F ()  
{  
    if sig_tok == '(' then  
    { print(7);  
      equipara ();  
      E;  
      equipara ());  
    }  
    else  
    { print(8);  
      equipara (id)  
    }  
}
```

NOTA. Los print son para obtener el parse, porque lo necesitáis para la práctica

## Ejemplo de un Recursivo Predictivo (con gramática EBNF)

```
Function Sent ()
{ if sig_tok == 'if' then
  { print(2);
    Sent_if
  }
  else if sig_tok == 'id' then
    { print(3);
      Sent_asig
    }
  else if sig_tok == 'begin' then
    { print(4);
      Bloque
    }
  else if sig_tok ∈ {;, end, else} /*
                                Follow(Sent) */
    then print(5)
    else error ()
}
```

Para la repetición (llaves EBNF)

```
Function Bloque ()
{ print(1);
  equipara (begin);
  Sent;
  while sig_tok == ';' do
  { equipara (;);
    Sent
  }
  equipara(end)
}
```

```
Function Sent_asig ()
{ print(7);
  equipara(id);
  equipara(:=);
  equipara(id);
  while sig_tok == '+' do
  { equipara(+);
    equipara(id);
  }
}
```

```
Function Cond ()
{
  print(8);
  equipara(id);
  equipara(>);
  equipara(id);
}
```

```
Function Sent_if ()
{ print(6);
  equipara (if);
  Cond;
  equipara (then);
  Sent;
  if sig_tok == 'else' then
  { equipara (else);
    Sent;
  }
}
```

Bloque → begin Sent {; Sent} end  
 Sent → Sent\_if | Sent\_asig | Bloque | λ  
 Sent\_if → if Cond then Sent [else Sent]  
 Sent\_asig → id := id {+ id}  
 Cond → id > id

NOTA. EBNF: { } repetición, [ ] opcionalidad

```
Function equipara (t)
{
  if sig_tok == t
  then sig_tok := ALEX()
  else error ()
}
```

```
Function A_Sint ()
{
  sig_tok := ALEX()
  E;
  if sig_tok ≠ '$' then error ()
}
```

Para la opcionalidad (corchetes EBNF)





# 1. Ejemplo de análisis de una cadena correcta $\omega = \text{id} + \text{id} \$$

A\_Sint ()

sig\_tok := 'id'

E ()

← regla 1

T ()

← regla 4

F ()

← regla 8

[equipara (id) /\* sig\_tok := '+'

T'() /\* sale por el else sin hacer nada

← regla 6

E'()

← regla 2

[equipara (+) /\* sig\_tok := 'id'

T ()

← regla 4

F ()

← regla 8

[equipara (id) /\* sig\_tok := '\$'

T'() /\* sale por el else sin hacer nada ← regla 6

E' () /\* sale por el else sin hacer nada

← regla 3

Se ha construido el árbol de la cadena completa (sig\_tok = '\$'), luego es CORRECTA

1.  $E \rightarrow T E'$
2.  $E' \rightarrow + T E'$
3.  $E' \rightarrow \lambda$
4.  $T \rightarrow F T'$
5.  $T' \rightarrow * F T'$
6.  $T' \rightarrow \lambda$
7.  $F \rightarrow ( E )$
8.  $F \rightarrow \text{id}$

Traza de  
ejecución del  
Descendente  
Recursivo

Parse: 148624863





## 2. Ejemplo de análisis de una cadena errónea

 $\omega =$  id + id ) \$

1.  $E \rightarrow T E'$
2.  $E' \rightarrow + T E'$
3.  $E' \rightarrow \lambda$
4.  $T \rightarrow F T'$
5.  $T' \rightarrow * F T'$
6.  $T' \rightarrow \lambda$
7.  $F \rightarrow ( E )$
8.  $F \rightarrow id$

A\_Sint ()

sig\_tok := 'id'

E ()

← regla 1

T ()

← regla 4

F()

← regla 8

[equipara (id) /\* sig\_tok := '+'

T'() /\* sale por el else sin hacer nada

← regla 6

E'()

← regla 2

[equipara (+) /\* sig\_tok := 'id'

T ()

← regla 4

F()

← regla 8

[equipara (id) /\* sig\_tok := ')

T'() /\* sale por el else sin hacer nada ← regla 6

E' () /\* sale por el else sin hacer nada

← regla 3

Se ha construido el árbol pero **no** de la cadena completa (sig\_tok ≠ '\$'), luego es ERRÓNEA

Traza de  
ejecución del  
Descendente  
Recursivo



### 3. Ejemplo de análisis de una cadena errónea

 $\omega =$  id + \* id \$

1.  $E \rightarrow T E'$
2.  $E' \rightarrow + T E'$
3.  $E' \rightarrow \lambda$
4.  $T \rightarrow F T'$
5.  $T' \rightarrow * F T'$
6.  $T' \rightarrow \lambda$
7.  $F \rightarrow ( E )$
8.  $F \rightarrow id$

A\_Sint ()

sig\_tok := 'id'

E ()

← regla 1

T ()

← regla 4

F()

← regla 8

[equipara (id) /\* sig\_tok := '+'

T'() /\* sale por el else sin hacer nada

← regla 6

E'()

← regla 2

[equipara (+) /\* sig\_tok := '\*'

T ()

← regla 4

F()

← regla 8

[equipara (id) /\* equipara detecta error ('\*' ≠ 'id')

Traza de  
ejecución del  
Descendente  
Recursivo

No se ha podido construir el árbol de la cadena, luego es ERRÓNEA