ANÁLISIS SINTÁCTICO DESCENDENTE PREDICTIVO CON GRAMÁTICAS LL(1)



Analizador Sintáctico Descendente Predictivo

- ¿Qué gramáticas vamos a usar?
 - \rightarrow LL(1)

factorizada

¿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)
- factorizaria Factorización por la izquierda
 - → Para cada No terminal A que no esté factorizado, encontrar el prefijo común más largo y "sacar factor común"

G:
$$A \rightarrow \alpha \beta_1 | \alpha \beta_2 | ... | \alpha \beta_n | \gamma$$

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

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





Ejemplo de factorización de una G





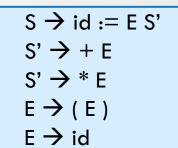
$$A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \mid ... \mid \alpha \beta_n \mid \gamma$$

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

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

$$S \rightarrow id := E + E$$

 $S \rightarrow id := E * E$
 $E \rightarrow (E)$
 $E \rightarrow id$





ANÁLISIS SINTÁCTICO DESCENDENTE CON GRAMÁTICAS LL(1)

ANALIZADOR SINTÁCTICO DESCENDENTE PREDICTIVO RECURSIVO

```
...
7. F → (E)
8. F → 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 \neq '\\$' then error ()
}





Ejemplo de construcción de un Recursivo Predictivo

```
Function A_Sint ()
{
    sig_tok := ALex()
    E;
    if sig_tok \neq '\( \frac{5}{2} \) 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 ()
}
```



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 \rightarrow begin Sent {; Sent} end

Sent \rightarrow Sent_if | Sent_asig | Bloque | \lambda

Sent_if \rightarrow if Cond then Sent [else Sent]

Sent_asig \rightarrow id := id {+ id}

Cond \rightarrow 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 \neq '\(\frac{5}{2}\) then error ()
}
```

Para la opcionalidad (corchetes EBNF)

Aurora Pérez Pérez





1. Ejemplo de análisis de una cadena correcta $\omega =$

```
\omega =  id + id $
```

```
A Sint ()
  sig tok := 'id'
  E ()
                                                       regla 1
                                                       regla 4
       F()
                                                      regla 8
         equipara (id) /* sig_tok := '+'
                 /* sale por el else sin hacer nada
       T'()
                                                      regla 6
     E'()
                                                      regla 2
                     /* sig tok := 'id'
       equipara (+)
       T ()
                                                       regla 4
          F()
                                                       regla 8
            _equipara (id) /* sig_tok := '$'
                    /* sale por el else sin hacer nada 🛑 regla 6
```

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

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

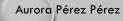
ejecución del Descendente

Parse: 148624863

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



regla 3







2. Ejemplo de análisis de una cadena errónea

```
id + id ) $
\omega =
```

```
A Sint ()
  sig_tok := 'id'
  E ()
                                                        regla 1
                                                       regla 4
       F()
                                                       regla 8
         equipara (id) /* sig_tok := '+'
                  /* sale por el else sin hacer nada
       T'()
                                                      regla 6
     E'()
                                                       regla 2
                     /* sig tok := 'id'
       equipara (+)
        T ()
                                                       regla 4
          F()
                                                       regla 8
            _equipara (id) /* sig_tok := ')'
                    /* sale por el else sin hacer nada 🛑 regla 6
       E'() /* sale por el else sin hacer nada
```

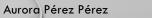
1. E \rightarrow T E' 2. E' \rightarrow + T E' 3. E' $\rightarrow \lambda$ $4. T \rightarrow F T'$ 5. T' \rightarrow * F T' 6. T' $\rightarrow \lambda$ $7. F \rightarrow (E)$ 8. $F \rightarrow id$

Traza de ejecución del Descendente Recursivo

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



regla 3







3. Ejemplo de análisis de una cadena errónea

$$\omega = \left[id + * id \right]$$
\$

```
A_Sint ()
  sig_tok := 'id'
  E ()
                                                       regla 1
                                                       regla 4
       F()
                                                       regla 8
         equipara (id) /* sig_tok := '+'
                  /* sale por el else sin hacer nada
                                                      regla 6
     E'()
                                                       regla 2
                     /* sig_tok := '*'
       equipara (+)
        T ()
                                                       regla 4
          F()
                                                       regla 8
            _equipara (id) /* equipara detecta error ('*' ≠ 'id')
```

1. E → T E'

2.
$$E' \rightarrow + T E'$$

3. E'
$$\rightarrow \lambda$$

4. T
$$\rightarrow$$
 F T'

6. T'
$$\rightarrow \lambda$$

$$7. F \rightarrow (E)$$

8. F
$$\rightarrow$$
 id

ejecución del Descendente

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

