Tema 4 – Análisis Sintáctico

Sesión 2: Análisis sintáctico descendente (I)

Sesión 2: Análisis sintáctico descendente (I)

- Análisis descendente:
  - Análisis descendente recursivo:
    - Con retroceso
    - Sin retroceso (predictivo)
  - Análisis descendente no recursivo predictivo (≡ tabular):
    - Análisis *LL(K)*
      - Análisis *LL(1)*
- Análisis ascendente:
  - Análisis ascendente con retroceso
  - Análisis de gramáticas de precedencia simple
  - Análisis de gramáticas de precedencia de operador
  - Análisis *LR(K)*
    - Análisis *LR(1)*
      - Análisis *SLR(1)*

Descenso recursivo con retroceso (1)

- Un analizador sintáctico recursivo con retroceso va seleccionando las reglas a aplicar para reconocer la entrada aplicando una estrategia de vuelta atrás (*backtracking*)
- Requisito (sobre la gramática):
  - Que no se puedan generar recursiones infinitas en la expansión de los no terminales a partir de la parte derecha de sus producciones asociadas
- Preproceso sobre la gramática para cubrir el requisito:
  - Eliminar la recursividad por la izquierda

Descenso recursivo con retroceso (2)

- Eliminación de la recursividad por la izquierda de una gramática:
  - ENTRADA: Una gramática, G= (N,T,S,P), con reglas agrupadas por metanociones y numeradas
  - SALIDA: Gramática G', equivalente a G, sin recursividad por la izquierda
  - PROCESO:
    - $i \leftarrow 1$; $n \leftarrow |N|$ (* |N| = número de metanociones de G *)
    - MIENTRAS $i \leq n$:
      - Sustituir las reglas de la forma:
        - $A_i \rightarrow A_i \alpha_1 \mid \ldots \mid A_i \alpha_p \mid \beta_1 \mid \ldots \mid \beta_q$

        por reglas de la forma:
        - $A_i \rightarrow (\beta_1 \mid \ldots \mid \beta_q) \{ \alpha_1 \mid \ldots \mid \alpha_p \}$
      - $i \leftarrow i + 1$;
  - OBSERVACIÓN: $\alpha_j \in (N \cup T)^*$; $\beta_k \in T^*$

Descenso recursivo con retroceso (3)

- Algoritmo de descenso recursivo con retroceso:
  - ENTRADA:
    - Gramática de contexto libre, no recursiva por la izquierda
    - Cadena de entrada: $w = a_1, a_2, ..., a_n$, $n \geq 0$
  - SALIDA:
    - Un *parse* a izquierdas para *w*, si existe; en otro caso, error
  - PROCESO:
    1. Ordena (agrupando por metanociones) todas y cada una de las reglas de la gramática
    2. Numera las reglas conforme al orden determinado
    3. NODO_ACTIVO ← S
       (* Comienza la generación del árbol con el axioma de la gramática (S) *)
    4. *Scan* (*token*)

Descenso recursivo con retroceso (4)

- PROCESO (cont.):

5. EJECUTA:
   - SI NODO_ACTIVO = *A* (es un no terminal) :
     - SI quedan reglas alternativas para expandir *A* :
       * Toma la primera alternativa para *A* y la expande, creando sus descendientes
       * NODO_ACTIVO ← descendiente de *A* más a la izquierda
     - EN OTRO CASO:
       - SI NODO_ACTIVO = S ⇒ ERROR
       - EN OTRO CASO: NODO_ACTIVO ← *Anterior* (NODO_ACTIVO)
         (* más a la izquierda y/o más arriba *)
   - SI NODO_ACTIVO = *a* (es un terminal) ∧ *Concuerda* (*token*, *a*):
     - NODO_ACTIVO ← siguiente pariente a la derecha de *a*
     - *Scan* (*token*)
   - EN OTRO CASO:
     - NODO_ACTIVO ← *Anterior* (NODO_ACTIVO)
     - Si es el caso, restaura *token(s)* a la entrada

   HASTA *Fin* (*w*) o ERROR
6. SI NO se ha podido llegar a construir el árbol: ERROR
7. EN OTRO CASO: El *parse* será la secuencia de los números de las reglas utilizadas

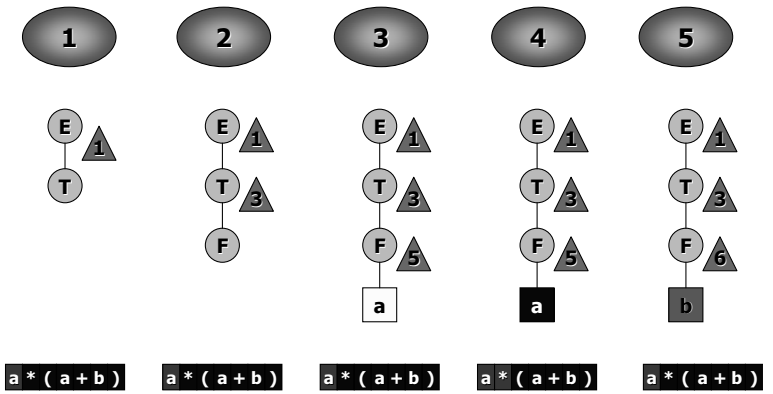
Descenso recursivo con retroceso: ejemplo

- Dada la gramática:
  1. E → T
  2. E → T+E
  3. T → F
  4. T → F*T
  5. F → a
  6. F → b
  7. F → (E)

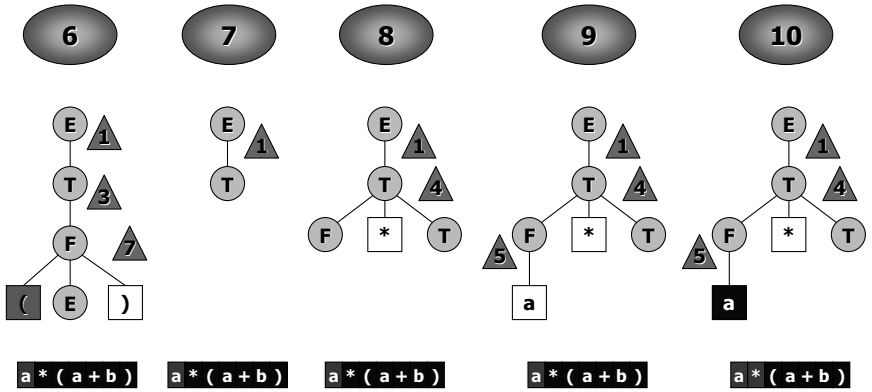
- Y dada la sentencia:
  a*(a+b)

- ¿Cuál es la traza de ejecución del algoritmo de descenso recursivo?

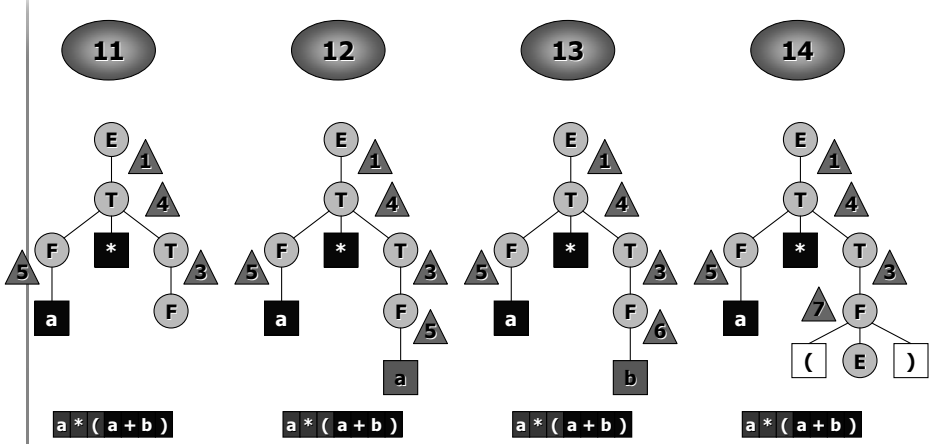
Descenso recursivo con retroceso: ejemplo (1)



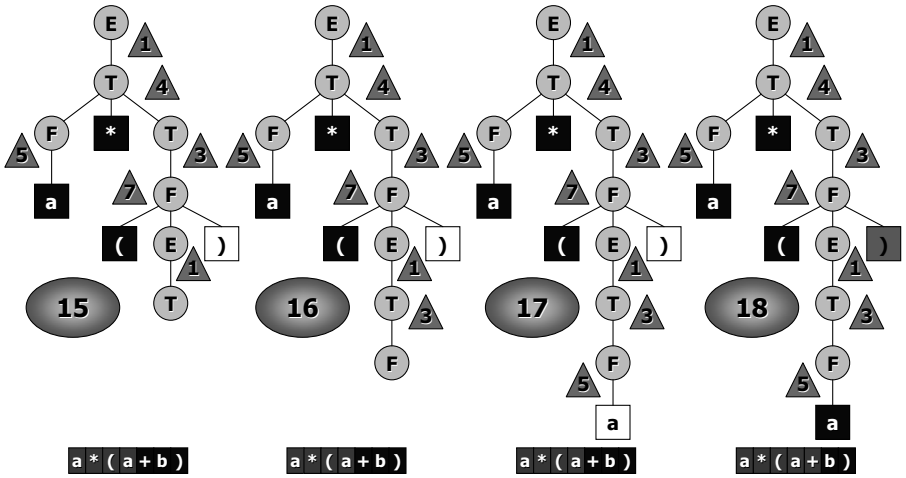
Descenso recursivo con retroceso: ejemplo (2)



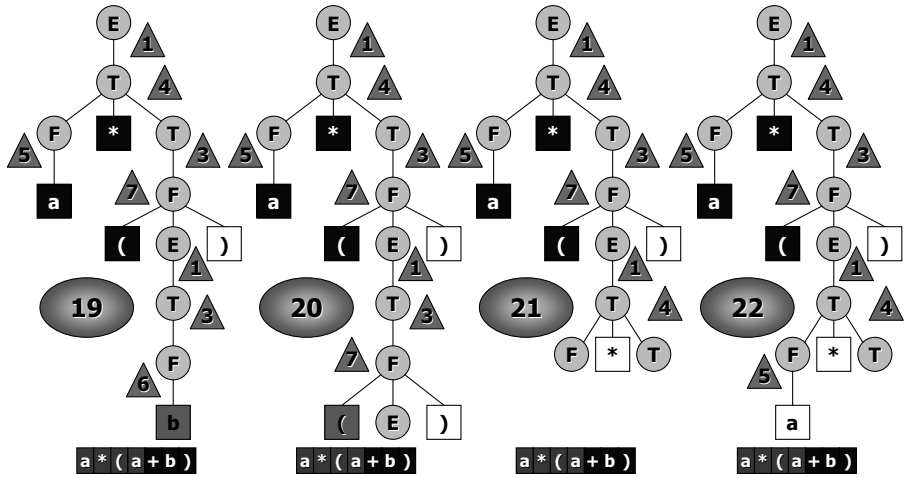
Descenso recursivo con retroceso: ejemplo (3)



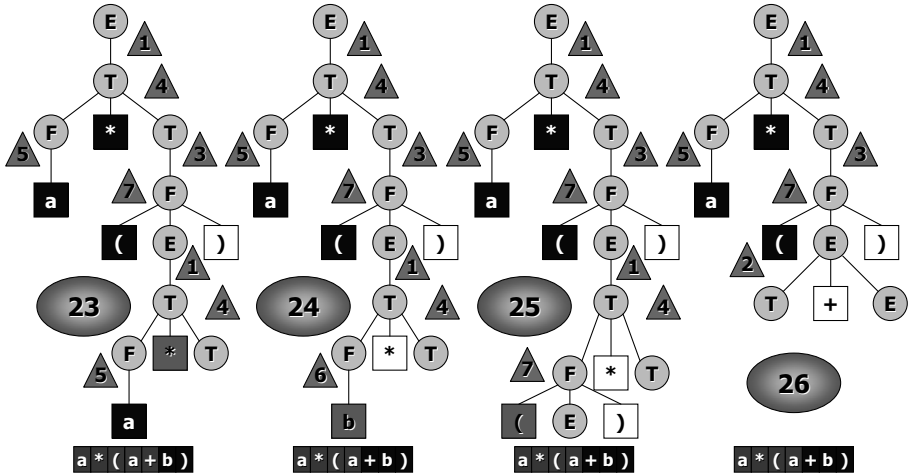
Descenso recursivo con retroceso: ejemplo (4)



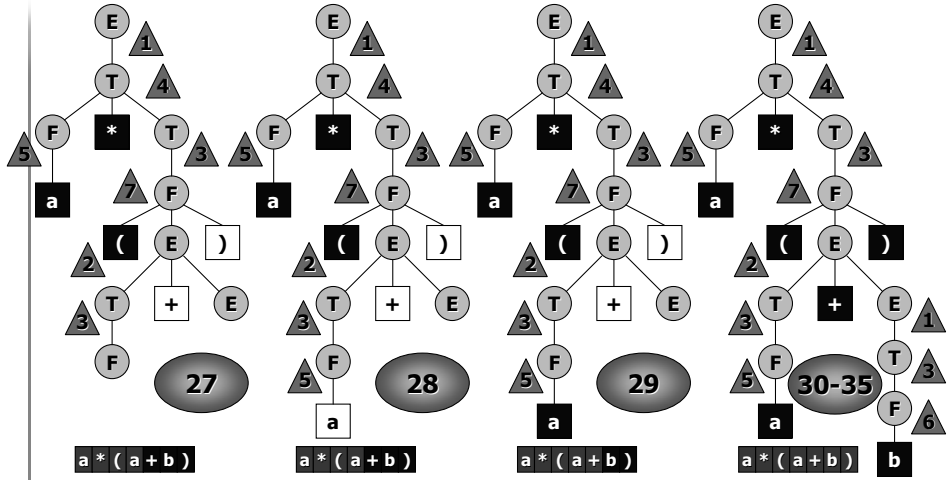
Descenso recursivo con retroceso: ejemplo (5)



Descenso recursivo con retroceso: ejemplo (6)



Descenso recursivo con retroceso: ejemplo (7)



Descenso recursivo predictivo (1)

- Un analizador sintáctico predictivo elimina la necesidad de realizar retrocesos
- Requisito (sobre la gramática):
  - En todo momento, a partir del siguiente *token* de entrada, la regla que se puede aplicar a continuación es única
- Preproceso sobre la gramática para cubrir el requisito:
  - Eliminar la recursividad por la izquierda
  - Factorización por la izquierda de la gramática

Descenso recursivo predictivo (2)

- Factorización por la izquierda de una gramática:
  - ENTRADA: Una gramática, G= (N,T,S,P)
  - SALIDA: Gramática G', factorización por la izquierda de G
  - PROCESO:
    - PARA CADA A $\in$ N :
      - REPETIR:
        - $\alpha \leftarrow$ PREFIJO_COMÚN_MÁS_LARGO(A)
        - SI $\alpha \neq \lambda$:
          * Reemplazar todas las producciones de la forma:
            – $A \rightarrow \alpha\ \beta_1 \mid \ldots \mid \alpha\ \beta_n \mid \gamma$
            por reglas de la forma:
            – $A \rightarrow \alpha\ A' \mid \gamma$
            – $A' \rightarrow \beta_1 \mid \ldots \mid \beta_n$

        MIENTRAS existan prefijos comunes para A.
  - OBSERVACIÓN: α , $\beta_k \in (N \cup T)^*$

Descenso recursivo predictivo (3)

- Implementación de un analizador sintáctico descendente recursivo predictivo:
  - Se asocia un procedimiento (o una función) a cada no terminal
  - En cada procedimiento (o función) se genera el código para concordar la parte derecha de las producciones (un bloque por cada símbolo):
    - SI el siguiente símbolo de la parte derecha es un terminal, *a*, el bloque asociado es:
      - SI *Concuerda* (*token*, *a*) :
        - Pide siguiente *token*
      - EN OTRO CASO :
        - ERROR (sintáctico)
    - EN OTRO CASO (* es un no terminal *A* *) :
      - El bloque asociado genera las llamadas a los procedimientos (o funciones) de la parte derecha de las producciones asociadas a *A*

Descenso recursivo predictivo: ejemplo (1)

- Dada la gramática:
  - $S \rightarrow$ if B then S | write B | i := B
  - $B \rightarrow i = i \mid i \neq i \mid$ true | false

¿Cuál es el código que implementa el analizador sintáctico recursivo predictivo para su reconocimiento?

Descenso recursivo predictivo: ejemplo (2)

```
PROCEDURE S (VAR token);
BEGIN
- IF (token.cod = ID) THEN
    BEGIN (* i := B *)
    - Scan (token);
    - IF (token.cod = OP_ASIG) THEN
        - Scan (token)
      ELSE
        - Error (sintáctico);
    - B (token);
    END; (* IF * i := B *)
  ELSIF ((token.cod = PAL_RES) AND (token.atr = IF_INDEX)) THEN
    BEGIN (* if B then S *)
    - Scan (token);
    - B (token);
```


Descenso recursivo predictivo: ejemplo (3)

```
    - IF ((token.cod = PAL_RES) AND (token.atr = THEN_INDEX)) THEN
        - Scan (token)
      ELSE
        - Error (sintáctico);
    - S (token);
    END; (* IF * if B then S *)
  ELSIF ((token.cod=PAL_RES) AND (token.atr=WRITE_INDEX)) THEN
    BEGIN (* write B *)
    - Scan (token);
    - B (token);
    END; (* IF * write B *)
  ELSE
    - Error (sintáctico);
  END (* IF *)
END; (* S *)
```

Descenso recursivo predictivo: ejemplo (4)

```
PROCEDURE B (VAR token);
BEGIN
■  IF (token.cod = ID) THEN
     BEGIN (* i = i | i ≠ i : NO FACTORIZADO por la IZQUIERDA, pero fácil de procesar *)
     – Scan (token);
     – IF ((token.cod = OP_COMP) AND (token.atr ∈ {IGUAL, DISTINTO})) THEN
         ■ Scan (token)
       ELSE
         ■ Error (sintáctico);
     – IF (token.cod = ID) THEN
         ■ Scan (token)
       ELSE
         ■ Error (sintáctico);
     END; (* IF * i = i | i ≠ i *)
   ELSIF ((token.cod=PAL_RES) AND (token.atr∈{TRUE_INDEX, FALSE_INDEX})) THEN
     – Scan (token);
   ELSE
     – Error (sintáctico);
   END (* IF*)
END; (* B *);
```


Descenso recursivo predictivo: ejercicios

1. Escribe el *parser* de descenso recursivo para la gramática de las expresiones aritméticas (expresada con reglas *EBNF*):
   a) $S \rightarrow E \$$
   b) $E \rightarrow [-] T \{ (+ | -) T \}$
   c) $T \rightarrow F \{ (* | /) F \}$
   d) $F \rightarrow id | (E)$

   NOTAS:
   - $ representa el fin de la cadena de entrada.
   - Observa que hay dos tipos diferentes de paréntesis en la descripción de la gramática, que hay que diferenciar al interpretar sus reglas.
2. Aplica el ejercicio anterior al reconocimiento de: (-a+b+c)*d$
3. Completa el descenso recursivo de los dos ejercicios anteriores incluyendo la sentencia de asignación, cambiando la producción del axioma (a) por: $S \rightarrow id := E \$$

Bibliografía

- Aho, A. V.; Sethi, R.; Ullman, J. D.: *Compilers: Principles, Techniques and Tools*. Massachusetts: Addison-Wesley Publishing Company, 1986.
- Alfonseca Cubero, E.; Alfonseca Moreno, M.; Moriyón Salomón, R. Teoría de autómatas y lenguajes formales. Madrid: Mc-Graw-Hill/Interamericana de España, S.A.U., 2007.
- Grogono, P. Programación en Pascal. Wilmington, Delaware (EE.UU.):Addison-Wesley Iberoamericana, 1996.
- Sanchís Llorca, F. J. y Galán Pascual, C. Compiladores: Teoría y construcción. Madrid: Editorial Paraninfo, 1986.