Tema 4 – Análisis Sintáctico

Sesiones 3, 4 y 5: Análisis sintáctico descendente (II)

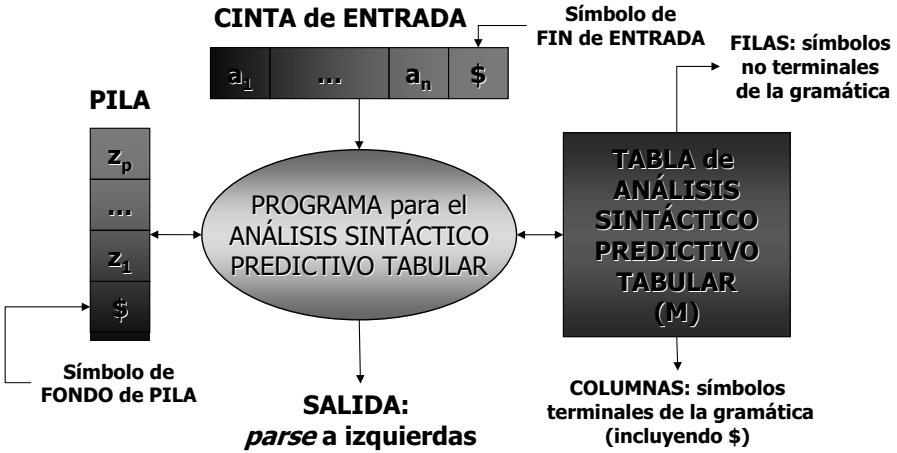
Sesiones 3, 4 y 5: Análisis sintáctico descendente predictivo (≡ tabular)

- Esquema de funcionamiento del descenso tabular
- Gramáticas *LL(1)*
  - Los conjuntos PRIMERO y SIGUIENTE
    - Algoritmo de cálculo del conjunto PRIMERO
      - Ejemplo de aplicación
    - Algoritmo de cálculo del conjunto SIGUIENTE
      - Ejemplo de aplicación
  - Condiciones *LL(1)*
    - Comprobación de las condiciones *LL(1)*
    - Ejemplo de aplicación
- Construcción de tablas para el descenso tabular
  - Ejemplo de aplicación
  - Determinación de casos de error
- Gestión de errores: introducción
- Algoritmo de descenso tabular
  - Ejemplo de aplicación
- Observaciones y anexos

Recordatorio: Tipos de analizadores sintácticos

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

An. descendente predictivo tabular: esquema de funcionamiento



Los conjuntos PRIMERO (X) y SIGUIENTE (X)

- Dada una gramática, G = (N, T, S, P), y una forma sentencial, $\alpha \in (N \cup T)^*$, al conjunto de aquellos elementos terminales que pueden aparecer como **primer símbolo** en cualquier cadena de L(G) derivada a partir de α se le llama **PRIMERO (α)**. Formalmente:
  - PRIMERO (α) = $\begin{cases} \text{Si } \alpha \Rightarrow^* \lambda & \{ a \in T \mid \alpha \Rightarrow^* a\beta \} \cup \{ \lambda \} \\ \text{E. O. C.} & \{ a \in T \mid \alpha \Rightarrow^* a\beta \} \end{cases}$
  - En el contexto apropiado, si un *token* a∈PRIMERO(α), esto indica que puede tomarse una **nueva regla** con α a la izquierda de la parte derecha para continuar el análisis.
- Dada una gramática, G = (N, T, S, P), y una metanoción de G, X ∈ N, al conjunto de aquellos elementos terminales que pueden aparecer como primer símbolo **a continuación** de cualquier cadena de L(G) derivada a partir de X se le llama **SIGUIENTE (X)**. Formalmente:
  - SIGUIENTE (X) = $\begin{cases} \text{Si } S \Rightarrow^+ \alpha X & \{ a \in T \mid S \Rightarrow^+ \alpha X a \beta \ (\alpha, \beta \in (N \cup T)^*) \} \cup \{ \$ \} \\ \text{E. O. C.} & \{ a \in T \mid S \Rightarrow^+ \alpha X a \beta \ (\alpha, \beta \in (N \cup T)^*) \} \end{cases}$
  - En el contexto apropiado, si un *token* a∈SIGUIENTE(X), esto indica que se han terminado de procesar los *tokens* en los que deriva X dentro del árbol sintáctico asociado a la entrada que se está analizando.

Cálculo de PRIMERO (X): algoritmo

- ENTRADA: Una gramática, G = (N, T, S, P) (no recursiva por la izquierda y factorizada por la izquierda).
- SALIDA: La lista de conjuntos PRIMERO (X), donde X ∈ N∪T.
- PROCESO:
  - ∀ X ∈ N∪T, PRIMERO (X) ← ∅
  - EJECUTA:
    - SI X ∈ T :
      - PRIMERO (X) ← PRIMERO (X) ∪ { X }

      EN OTRO CASO (* X ∈ N *) :
      - SI (X → λ) ∈ P :
        - PRIMERO (X) ← PRIMERO (X) ∪ { λ }
      - SI (X → Y_1 … Y_k) ∈ P (* Y_i ∈ N∪T *):
        - ∀ j ∈ {1, …, k} : SI λ ∈ PRIMERO (Y_1) ∩ … ∩ PRIMERO (Y_{j-1})
          - PRIMERO (X) ← PRIMERO (X) ∪ (PRIMERO (Y_j) \ { λ })
        - SI λ ∈ PRIMERO (Y_1) ∩ … ∩ PRIMERO (Y_k)
          - PRIMERO (X) ← PRIMERO (X) ∪ { λ }

    MIENTRAS ∃ X ∈ N∪T,, PRIMERO (X) cambie.

Esta condición SE CUMPLE de forma trivial para *j = 1*

Cálculo de PRIMERO (X): ejemplo (1)

- Obtener los conjuntos PRIMERO (X) asociados a la gramática:

1. E → T
2. E → E**+**T
3. T → F
4. T → T***F
5. F → **(** E **)**
6. F → **id**

PASO 0.1: Eliminación de la recursividad por la izquierda:

1. E → E+T | T ⇒ { – E → TE′ ; – E′ → +TE′ | λ }
2. T → T*F | F ⇒ { – T → FT′ ; – T′ → *FT′ | λ }
3. F → (E)
4. F → id

PASO 0.2: Factorización por la izquierda: HECHO

Cálculo de PRIMERO (X): ejemplo (2)

- PRIMERO (**+**) = {**+**}
- PRIMERO (*) = {*}
- PRIMERO (**(**) = {**(**}
- PRIMERO (**)**) = {**)**}
- PRIMERO (**id**) = {**id**}

1. E → TE′
2. E′ → **+**TE′ | λ
3. T → FT′
4. T′ → ***FT′ | λ
5. F → **(** E **) | id**

- PRIMERO (F) = ∅ ∪ (PRIMERO (**(**) \ {λ}) ∪ (PRIMERO (**id**) \ {λ}) = = {**(**} ∪ {**id**} = {**(**, **id**}
- PRIMERO (T) = ∅ ∪ (PRIMERO (F) \ {λ}) = {**(**, **id**}
- PRIMERO (E) = ∅ ∪ (PRIMERO (T) \ {λ}) = {**(**, **id**}
- PRIMERO (T′) = ∅ ∪ (PRIMERO (*) \ {λ}) ∪ {λ} = {*} ∪ {λ} = {*, λ}
- PRIMERO (E′) = ∅ ∪ (PRIMERO (**+**) \ {λ}) ∪ {λ} = {**+**} ∪ {λ} = {**+**, λ}

Cálculo de SIGUIENTE (X): algoritmo

- ENTRADA: Una gramática, G = (N, T, S, P) (no recursiva por la izquierda y factorizada por la izquierda).
- SALIDA: La lista de conjuntos SIGUIENTE (X), donde $X \in N$.
- PROCESO:
  - $\forall X \in N \setminus \{ S \}$, SIGUIENTE (X) $\leftarrow \varnothing$
  - SIGUIENTE (S) $\leftarrow \{ \$ \}$
  - EJECUTA:
    - SI $(X \rightarrow \alpha Y \beta) \in P$ (* $\alpha, \beta \in (N \cup T)^*$ *) :
      - SIGUIENTE (Y) $\leftarrow$ SIGUIENTE (Y) $\cup$ (PRIMERO (β) $\setminus \{ \lambda \}$)
      - SI (($\beta = \lambda$) $\vee$ ($\lambda \in$ PRIMERO (β))):
        - SIGUIENTE (Y) $\leftarrow$ SIGUIENTE (Y) $\cup$ SIGUIENTE (X)

  MIENTRAS $\exists X \in N$,, SIGUIENTE (X) cambie.

Cálculo de SIGUIENTE (X): ejemplo (1)

- Obtener los conjuntos SIGUIENTE (X) asociados a la gramática:
  1. $E \rightarrow TE'$
  2. $E' \rightarrow$ **+**$TE' \mid \lambda$
  3. $T \rightarrow FT'$
  4. $T' \rightarrow$ ***** $FT' \mid \lambda$
  5. $F \rightarrow$ **(** E **) | id**

- SIGUIENTE (E) = {**$**} $\cup$ (PRIMERO (**)**) – {λ}) = {**$**} $\cup$ {**)**} = {**$,)**}.
- SIGUIENTE (E') = $\varnothing \cup$ SIGUIENTE (E) = {**$,)**}.
- SIGUIENTE (T) = $\varnothing \cup$ (PRIMERO (E') $\setminus$ {λ}) $\cup$ SIGUIENTE (E) $\cup$ SIGUIENTE (E') = $\varnothing \cup$ ({**+,** λ} $\setminus$ {λ}) $\cup$ SIGUIENTE (E) = {**+**} $\cup$ {**$,)**} = {**+, $,)**}.
- SIGUIENTE (T') = $\varnothing \cup$ SIGUIENTE (T) = {**+, $,)**}.
- SIGUIENTE (F) = $\varnothing \cup$ (PRIMERO (T') $\setminus$ {λ}) $\cup$ SIGUIENTE (T) $\cup$ SIGUIENTE (T') = ({*, λ} $\setminus$ {λ}) $\cup$ SIGUIENTE (T) = {*} $\cup$ {**+, $,)**} = **{*, +, $,)**}.

Gramáticas *LL*(1): definición

- Se dice que una gramática, G = (N, T, S, P), es de tipo *LL*(1) si, $\forall C \subseteq P$, $C = \{A \rightarrow \alpha_i \in P\}_{i=1}^{k}$ se verifican las condiciones:
  - i. $\forall i, j \in \{1,\ldots,k\}, i \neq j,$
    $$PRIMERO(\alpha_i) \cap PRIMERO(\alpha_j) = \varnothing$$
  - ii. Si $A \rightarrow \lambda \in C$ ($\lambda = \alpha_1$):
    - $\forall i \in \{2, \ldots, k\}$ $PRIMERO(\alpha_i) \cap SIGUIENTE(A) = \varnothing$
- Las dos condiciones anteriores hacen que sea posible determinar sin duda alguna la siguiente regla a aplicar en análisis descendente:
  - Cada primer terminal derivable a partir de A proviene de una parte derecha distinta.
  - Además, si A tiene asociada una regla borradora, tampoco el siguiente símbolo después de A crea conflictos.

Gramáticas *LL*(1): ejemplo

- ¿Es la siguiente gramática de tipo *LL*(1)?
  1. $E \rightarrow TE'$
  2. $E' \rightarrow$ **+**$TE' \mid \lambda$
  3. $T \rightarrow FT'$
  4. $T' \rightarrow$ ***** $FT' \mid \lambda$
  5. $F \rightarrow$ **(** E **) | id**

  Las metanociones sin partes derechas alternativas no plantean problemas

  - Reglas conflictivas (a priori):
    - $E' \rightarrow$ **+**$TE' \mid \lambda$
      - PRIMERO (**+**TE') $\cap$ SIGUIENTE (E') = { **+** } $\cap$ { **$,)** } = $\varnothing$
    - $T' \rightarrow$ ***** $FT' \mid \lambda$
      - PRIMERO (***** FT') $\cap$ SIGUIENTE (T') = { ***** } $\cap$ { **+, $,)** } = $\varnothing$
    - $F \rightarrow$ **(** E **) | id**
      - PRIMERO (**(** E **)**) $\cap$ PRIMERO (**id**) = { **(** } $\cap$ { **id** } = $\varnothing$
- Por lo tanto, la gramática es de tipo *LL*(1)

Creación de tablas para el análisis descendente: algoritmo

- ENTRADA: Una gramática, G = (N, T, S, P) (no recursiva por la izquierda y factorizada por la izquierda).
- SALIDA: La tabla, M, de análisis descendente para G.
- PROCESO:
  - $\forall$ (X, t) $\in$ N $\times$ (T$\cup${\$}), M [X, t] $\leftarrow \varnothing$ (* Cada celda es un conjunto *)
  - $\forall$ (X $\rightarrow \alpha$) $\in$ P :
    - $\forall$ a $\in$ PRIMERO (α) :
      - M [X, a] $\leftarrow$ M [X, a] $\cup$ { X $\rightarrow \alpha$ }
      - SI $\lambda \in$ PRIMERO (α) :
        - $\forall$ b $\in$ SIGUIENTE (X) :
          - M [X, b] $\leftarrow$ M [X, b] $\cup$ { X $\rightarrow \alpha$ }
  - $\forall$ (X, t) $\in$ N $\times$ (T$\cup${\$}) :
    - SI M [X, t] = $\varnothing$:
      - M [X, t] $\leftarrow$ ERROR(sintáctico)

Creación de tablas para el análisis descendente: ejemplo (1)

- ¿Cuál es la tabla de análisis sintáctico descendente para esta gramática?
  1. E $\rightarrow$ TE′
  2. E′ $\rightarrow$ **+**TE′ | λ
  3. T $\rightarrow$ FT′
  4. T′ $\rightarrow$ ***FT′ | λ
  5. F $\rightarrow$ **(** E **) | id**

- PRIMERO (F) = {(, id} SIGUIENTE (F) = {*, +, $,)}
- PRIMERO (T) = {(, id} SIGUIENTE (T) = {+, $,)}
- PRIMERO (E) = {(, id} SIGUIENTE (E) = {$,)}
- PRIMERO (T′) = {*, λ} SIGUIENTE (T′) = {+, $,)}
- PRIMERO (E′) = {+, λ} SIGUIENTE (E′) = {$,)}
- PRIMERO (FT′) = PRIMERO (F) = {(, id}
- PRIMERO (TE′) = PRIMERO (T) = PRIMERO (FT′) = {(, id}

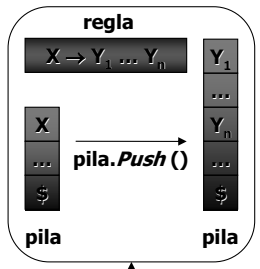
	+	*	(	)	id	$
E			E $\rightarrow$ TE′		E $\rightarrow$ TE′	
E′	E′ $\rightarrow$ +TE′			E′ $\rightarrow \lambda$		E′ $\rightarrow \lambda$
T			T $\rightarrow$ FT′		T $\rightarrow$ FT′	
T′	T′ $\rightarrow \lambda$	T′ $\rightarrow$ *FT′		T′ $\rightarrow \lambda$		T′ $\rightarrow \lambda$
F			F $\rightarrow$ (E)		F $\rightarrow$ id	

Análisis descendente predictivo tabular: observaciones

- Una forma alternativa de comprobar que una gramática es de tipo *LL*(1):
  - Se crea su tabla de análisis descendente
  - Si ninguna casilla de la tabla tiene más de una regla, la gramática es *LL*(1) – y no lo es en caso contrario
- En principio, no es necesario comprobar que una gramática esté limpia, sin recursividad por la izquierda y factorizada por la izquierda para hallar la tabla de análisis descendente:
  - Si es *LL*(1), se creará la tabla sin ambigüedades de todas formas
  - Pero manipular la gramática:
    - Facilita la aplicación de todos los algoritmos implicados
    - Puede llevar a la gramática *LL*(1) equivalente a la de partida
- Una gramática limpia:
  - Sin símbolos no terminables (véase anexo 1)
  - Sin símbolos no accesibles (véase anexo 2)

Análisis descendente predictivo tabular: algoritmo

- ENTRADA:
  - Gramática *G* = (*N*,*T*,*S*,*P*), no recursiva por la izquierda y factorizada por la izquierda.
  - *M*, tabla de análisis descendente para *G*.
  - *w* = $w_1 \ldots w_n \in$ T*, cadena de *tokens* a analizar.
- SALIDA:
  - *Parse* a izquierdas de *w*.
- PROCESO:
  - pila.*Vaciar* (); pilaParse.*Vaciar* (); pila.*Push* ($); pila.*Push* (S);
  - *Scan* (*token*);
  - MIENTRAS NO(pila.*Vacía* ()) :
    - SI pila.*Top* () $\in$ *T* $\cup$ {$} :
      - SI pila.*Top* () = *token* :
        - pila.*Pop* ();
        - SI *token* $\neq$ $:
          - *Scan* (*token*);
        - EN OTRO CASO : ÉXITO(Invierte(pilaParse));
      - EN OTRO CASO: ERROR (sintáctico);
    - EN OTRO CASO (* pila.*Top* () $\in$ *N* *) :
      - regla $\leftarrow$ M [pila.*Top* () , *token*];
      - SI regla = $\varnothing$: ERROR (sintáctico);
      - EN OTRO CASO:
        - pila.*Pop* (); pila.*Push* (Invierte(LadoDerecho(regla)));
        - pilaParse.*Push* (regla.Numero());

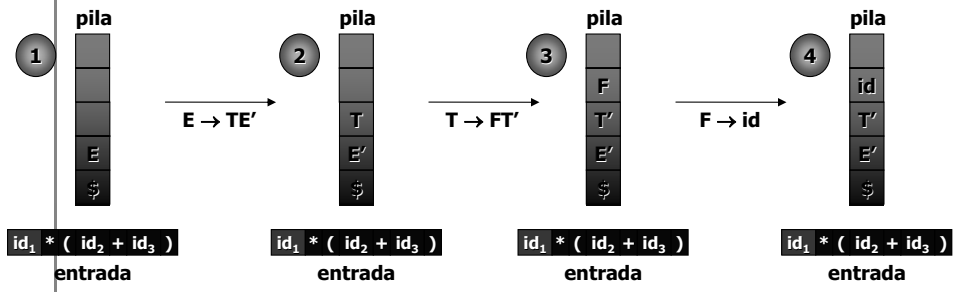


Análisis descendente predictivo tabular: traza del algoritmo (1)

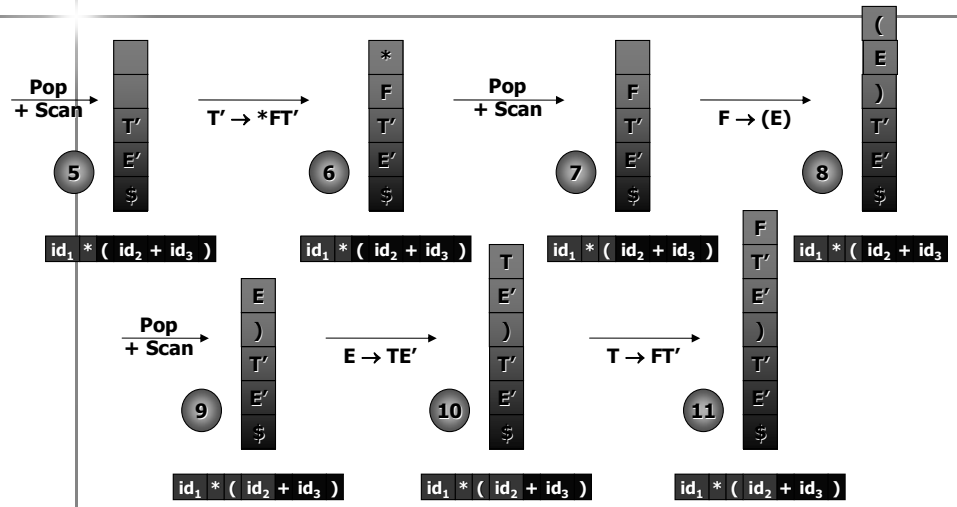
- Reconocimiento de la cadena

  $id_1 * (id_2 + id_3)$ \$

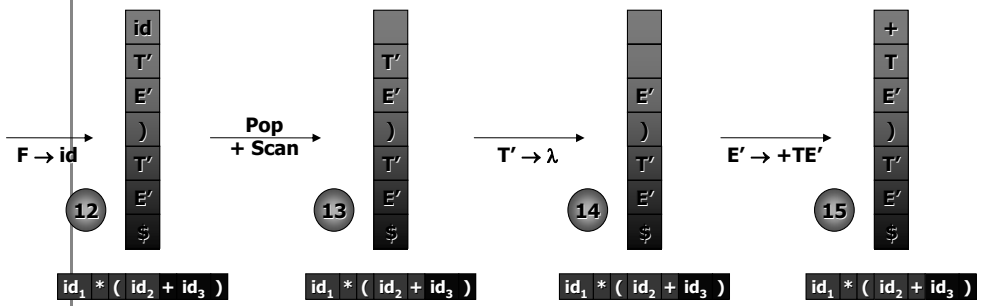
  con la tabla determinada anteriormente:



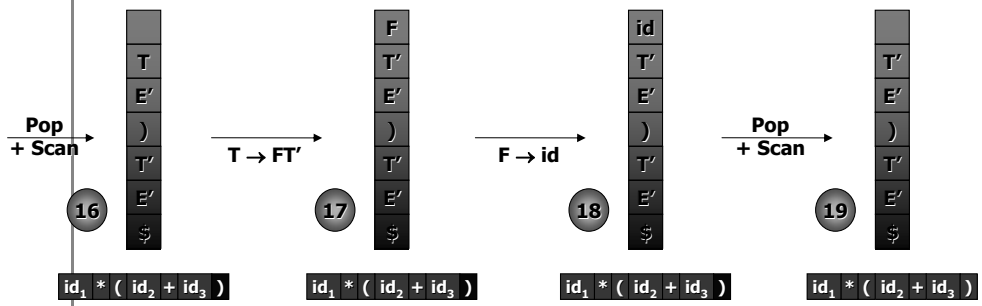
Análisis descendente predictivo tabular: traza del algoritmo (2)



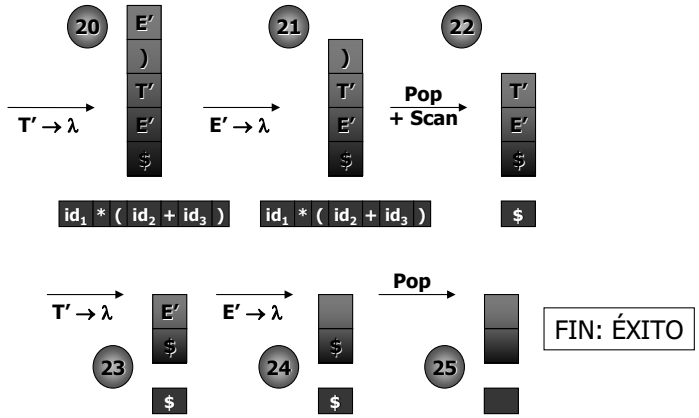
Análisis descendente predictivo tabular: traza del algoritmo (3)



Análisis descendente predictivo tabular: traza del algoritmo (4)



Análisis descendente predictivo tabular: traza del algoritmo (5)



Creación de tablas para el análisis descendente: mensajes de error

	+	*	(	)	id	$
E	EF1 / EC1	EF1 / EC2	E → TE′	EF1 / EC4	E → TE′	EF1 / EC6
E′	E′ → +TE′	EF2 / EC2	EF2 / EC3	E′ → λ	EF2 / EC5	E′ → λ
T	EF3 / EC1	EF3 / EC2	T → FT′	EF3 / EC4	T → FT′	EF3 / EC6
T′	T′ → λ	T′ → *FT′	EF4 / EC3	T′ → λ	EF4 / EC5	T′ → λ
F	EF5 / EC1	EF5 / EC2	F → (E)	EF5 / EC4	F → id	EF5 / EC6

- EF1: Error sintáctico: se esperaba '(' o identificador
- EF2: Error sintáctico: se esperaba '+', ')' o EOF
- EF3: Error sintáctico: se esperaba '(' o identificador
- EF4: Error sintáctico: se esperaba '+', '*', ')' o EOF
- EF5: Error sintáctico: se esperaba '(' o identificador

- EC1: Error sintáctico: símbolo inesperado ('+')
- EC2: Error sintáctico: símbolo inesperado ('*')
- EC3: Error sintáctico: símbolo inesperado ('(')
- EC4: Error sintáctico: símbolo inesperado (')')
- EC5: Error sintáctico: símbolo inesperado (id)
- EC5: Error sintáctico: fin de fichero inesperado

Gestión de errores: introducción (1)

- ¿Cuales son los **objetivos** de la gestión de errores?
  - Dar mensajes precisos y explicativos de cada error
  - No abortar el proceso de compilación si no es absolutamente necesario
  - Informar de las acciones tomadas para continuar la compilación en presencia de errores
- ¿Cómo continuar la compilación a pesar de la existencia de errores?
  - Con reglas heurísticas: no hay un método universal
    - **Recuperación**:
      - Se resincroniza la entrada con el estado del traductor (finito, en el caso del escáner; a pila, en el caso del an. sintáctico)
    - **Reparación**:
      - Se intenta diagnosticar y tratar el problema en la entrada

Gestión de errores: introducción (2)

- Con una **recuperación**:
  - **Modo pánico** (resincronización):
    - Se eliminan (o se saltan) símbolos de la entrada hasta encontrar alguno específico del **conjunto de resincronización** (determinado a priori)
      - En análisis sintáctico, normalmente, está formado por las palabras clave y los separadores (signos de puntuación)
    - Se manipula la pila (lo más complicado) para intentar que adopte un estado estable con la resincronización de la entrada

Gestión de errores: introducción (3)

- Con una **reparación**:
  - La entrada está en el estado α **t** β:
    - α es la parte de la cadena de entrada ya analizada
    - **t** es el símbolo de la entrada que se tiene que procesar en ese momento (el siguiente *token*, en el análisis sintáctico)
  - Hay tres posibles estrategias de actuación:
    - **Sustituir t** por otro símbolo, **t'**, más ajustado al estado de análisis del compilador, esperando que α **t'** β pertenezca al lenguaje
    - **Insertar** una subcadena γ por delante de **t**, esperando que α γ **t** β pertenezca al lenguaje
    - **Eliminar t**, esperando que α β pertenezca al lenguaje
  - Estrategias de generación de **t'** y γ:
    - Si hay un terminal, $\mathbf{t_p}$, como cima de la pila:
      - Usar $\mathbf{t_p}$ en lugar de **t** para continuar el análisis
    - Si hay un no terminal, **X**, como cima de la pila:
      - Usar, por orden, los terminales $\mathbf{t_i}$ para los que la celda de la tabla de análisis M [**X**, $\mathbf{t_i}$] no es un caso de error

Análisis descendente predictivo tabular: ejercicio 1 (1)

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

  comprueba si es de tipo *LL*(1).

Análisis descendente predictivo tabular: ejercicio 2 (1)

- Dada la gramática:
  - P $\rightarrow$ if E then P P' | write E
  - P' $\rightarrow$ else P | λ
  - E $\rightarrow$ id

  comprueba si es de tipo *LL*(1).

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 (USA): Addison-Wesley Iberoamericana, 1996.
- Sanchís Llorca, F. J. y Galán Pascual, C. Compiladores: Teoría y construcción. Madrid: Editorial Paraninfo, 1986.

Anexo 1: algoritmo de eliminación de símbolos no terminables

- ENTRADA: Una gramática, G = (N, T, S, P).
- SALIDA: Una gramática, G', equivalente a G, pero sin símbolos no terminables.
- PROCESO:
  - VIEJO $\leftarrow \varnothing$
  - NUEVO $\leftarrow \{ A \in N \mid (A \rightarrow t) \wedge (t \in T^*) \}$
  - MIENTRAS (VIEJO $\neq$ NUEVO) :
    - VIEJO $\leftarrow$ NUEVO
    - AÑADIDOS $\leftarrow \{ A \in N \mid (A \rightarrow \alpha) \wedge (\alpha \in (T \cup VIEJO)^*) \}$
    - NUEVO $\leftarrow$ VIEJO $\cup$ AÑADIDOS
  - TERMINABLES $\leftarrow$ VIEJO

Anexo 2: algoritmo de eliminación de símbolos no accesibles

- ENTRADA: Una gramática, G = (N, T, S, P).
- SALIDA: Una gramática, G', equivalente a G, pero sin símbolos no accesibles.
- REQUISITO: $\alpha, \beta \in (N \cup T)^*$
- PROCESO:
  - VIEJO $\leftarrow \{ S \}$
  - NUEVO $\leftarrow \{ X \in (N \cup T) \mid S \rightarrow \alpha X \beta \} \cup$ VIEJO
  - MIENTRAS (VIEJO $\neq$ NUEVO) :
    - VIEJO $\leftarrow$ NUEVO
    - AÑADIDOS $\leftarrow \{ B \in (N \cup T) \mid (A \rightarrow \alpha B \beta) \wedge (A \in VIEJO) \}$
    - NUEVO $\leftarrow$ VIEJO $\cup$ AÑADIDOS
  - ACCESIBLES $\leftarrow$ VIEJO