Universidad de Huelva

Tema 4

Análisis sintáctico ascendente

Índice

4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

4.6 Gestión de errores

4.7 Clasificación de gramáticas

Índice

4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

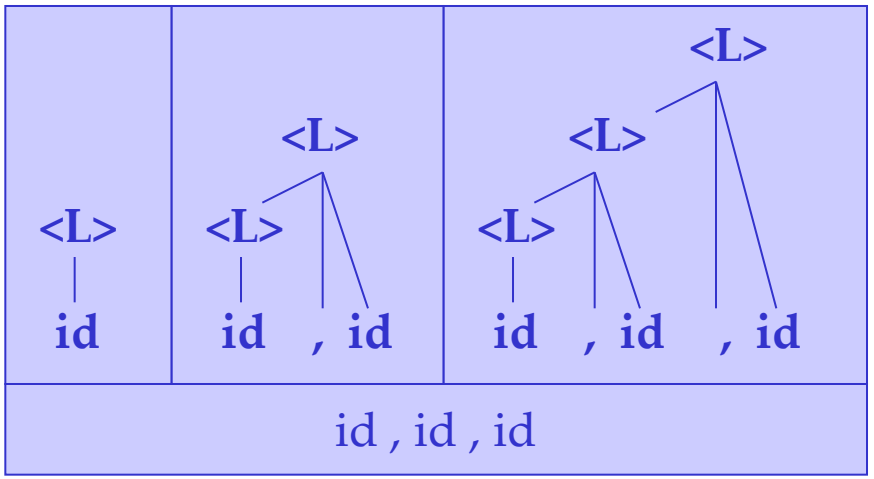
4.6 Gestión de errores

4.7 Clasificación de gramáticas

- El análisis sintáctico ascendente construye la inversa de la derivación por la derecha.
- Construye el árbol de análisis sintáctico de las hojas a la raíz.

Gramática

<L> → id
<L> → <L> , id



- El problema fundamental es decidir cuando lo que parece ser la parte derecha de una regla puede ser sustituida por la parte izquierda.
- No es un problema trivial ya que pueden existir ocasiones en las que es posible sustituir dos producciones diferentes.
- El conjunto de gramáticas que pueden ser analizadas mediante un análisis ascendente lineal se denomina LR(1).
- El conjunto LR(1) es mucho más amplio que el de las gramáticas LL(1).

Índice

4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

4.6 Gestión de errores

4.7 Clasificación de gramáticas

- El análisis ascendente lineal más utilizado es el ***algoritmo de desplazamiento-reducción*** (***shift-reduce***).
- Este algoritmo se basa en una pila de estados y una tabla de análisis.
- Existen diferentes métodos para generar la tabla de análisis (LR(0), SLR, LALR, LR(1)).
- El método LR(1) genera la tabla para cualquier gramática LR(1), pero genera tablas muy grandes.
- El método SLR genera tablas muy compactas pero no puede aplicarse a todas las gramáticas LR(1).
- El método LALR genera tablas compactas y puede aplicarse a la mayoría de gramáticas LR(1).

- Utiliza dos acciones básicas:
  - Desplazar: consiste en consumir un token de la cadena de entrada
  - Reducir: consiste en sustituir en la pila los símbolos de una parte derecha de una regla por su parte izquierda
- El algoritmo utiliza una pila de estados y una tabla de análisis con dos partes:
  - Acciones: para cada símbolo terminal y $.
  - Ir-a: para cada símbolo no terminal
  - Las filas corresponden a los estados

- Las acciones pueden ser
  - dj : desplazar y apilar el estado j
  - rk: reducir por la regla k-ésima.
    - Consiste en eliminar tantos estados de la pila como elementos en la parte derecha de la regla.
    - A continuación se analiza el estado de la cima de la pila p.
    - Por último se apila el estado Ir_a(p,A) siendo A el símbolo no terminal que define la regla k-ésima
  - aceptar: terminar el análisis aceptando la cadena
  - error: producir un error (cuando no se encuentra ninguna acción)

1: <S> → <B> <A> **end**
2: <A> → **begin** <C>
3: <C> → **codigo**
4: <B> → **tipo**
5: <B> → **id** <B>

Estado	**end**	**begin**	**codigo**	**tipo**	**id**	**$**	S	A	B	C
0				d3	d4		1		2	
1						aceptar				
2		d6						5		
3		r4								
4				d3	d4				7	
5	d8									
6			d10							9
7		r5								
8						r1				
9	r2									
10	r3									

PILA	ENTRADA	ACCIÓN
0	**id tipo begin codigo end $**	d4
0 4	**tipo begin codigo end $**	d3
0 4 3	**begin codigo end $**	r4 (<B> → **tipo**)
0 4 7	**begin codigo end $**	r5 (<B> → **id** <B>)
0 2	**begin codigo end $**	d6
0 2 6	**codigo end $**	d10
0 2 6 10	**end $**	r3 (<C> → **codigo**)
0 2 6 9	**end $**	r2 (<A> → **begin** <C>)
0 2 5	**end $**	d8
0 2 5 8	**$**	r1 (<S> → <B> <A> **end**)
0 1	**$**	aceptar

Índice

4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

4.6 Gestión de errores

4.7 Clasificación de gramáticas

- Elemento:
  - Se obtiene colocando un símbolo "·" en cualquier posición de la parte derecha de una regla
  - $<A> \rightarrow X_1 X_2 \ldots X_i \cdot X_{i+1} \ldots X_n$
  - Significa que los símbolos a la izquierda del punto ya han sido reconocidos
  - Las reglas vacías ($<A> \rightarrow \lambda$) sólo generan un elemento ($<A> \rightarrow \cdot$)

- Ejemplo de elementos:
  - Gramática:
    - <S> → <B> <A> **end**
    - <A> → **begin** <C>
    - <C> → **codigo**
    - <B> → **tipo**
    - <B> → **id** <B>
  - Elementos:

<S> → · <B> <A> **end**
<S> → <B> · <A> **end**
<S> → <B> <A> · **end**
<S> → <B> <A> **end** ·
<A> → · **begin** <C>

<A> → **begin** · <C>
<A> → **begin** <C> ·
<C> → · **codigo**
<C> → **codigo** ·
<B> → · **tipo**

<B> → **tipo** ·
<B> → · **id** <B>
<B> → **id** · <B>
<B> → **id** <B> ·

- Clausura de un conjunto de elementos:
  - Sea *I* un conjunto de elementos
  - Todos los elementos de *I* pertenencen a *clausura(I)*
  - Si clausura de *I* contiene un elemento de la forma
    - $<A> \rightarrow \alpha \cdot <B> \beta$
  - Entonces se añade a *clausura(I)* todos los elementos de la forma
    - $<B> \rightarrow \cdot \alpha$

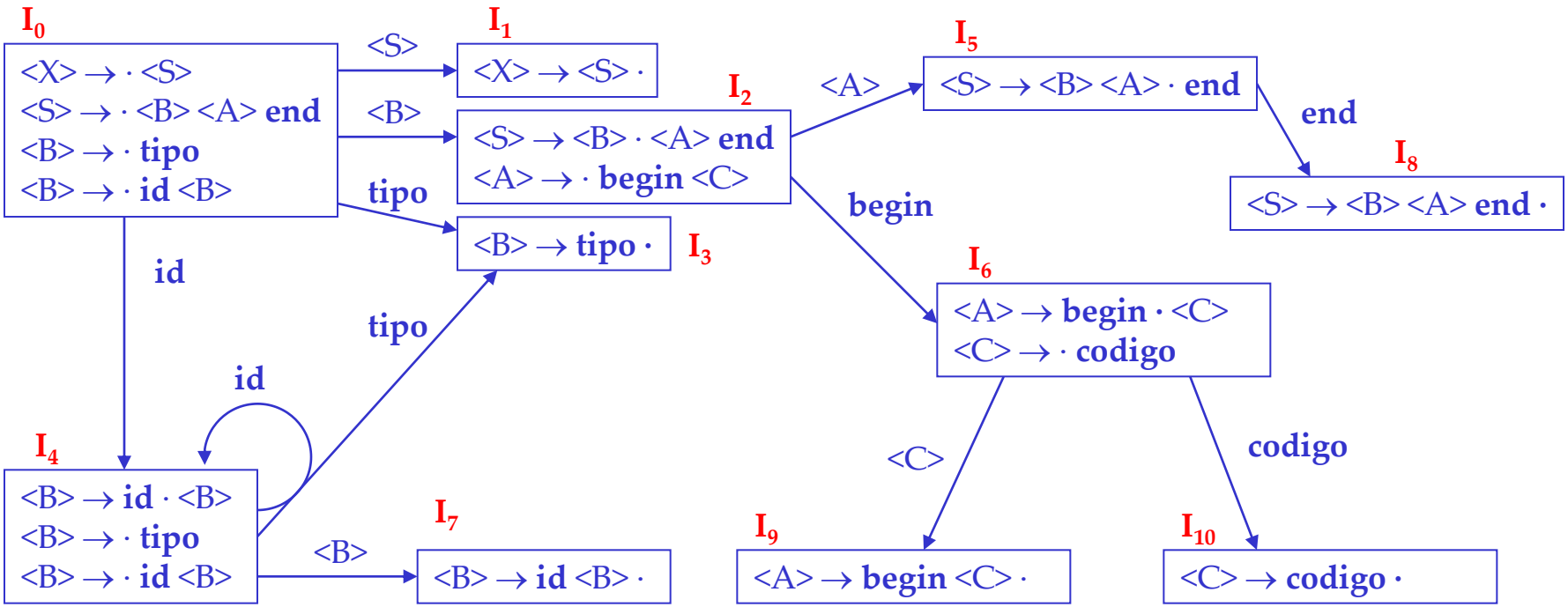
- Ejemplo de clausura de un conjunto de elementos:
  - Sea I el conjunto
    - <S> $\rightarrow \cdot$ <B> <A> **end**
  - *Clausura(I)* será el conjunto
    - <S> $\rightarrow \cdot$ <B> <A> **end**
    - <B> $\rightarrow \cdot$ **tipo**
    - <B> $\rightarrow \cdot$ **id** <B>

- Operación *Ir_a (GoTo)*
  - Sea *I* un conjunto de elementos y *A* un símbolo de la gramática (terminal o no terminal)
  - La operación *Ir_a(I,A)* da como resultado otro conjunto de elementos.
  - Para cada elemento de I de la forma
    - $<B> \rightarrow \alpha \cdot A \beta$
  - Se añade a *Ir_a(I,A)* los elementos del conjunto
    - clausura($<B> \rightarrow \alpha A \cdot \beta$)

- Ejemplo de operación *Ir_a*
  - Sea la gramática:
    - <S> → <B> <A> **end**
    - <A> → **begin** <C>
    - <C> → **codigo**
    - <B> → **tipo**
    - <B> → **id** <B>
  - Sea *I* el conjunto
    - <S> → · <B> <A> **end**
    - <B> → · **tipo**
    - <B> → **id** · <B>
  - El conjunto *Ir_a(I, <B>)* es
    - <S> → <B> · <A> **end**
    - <A> → · **begin** <C>
    - <B> → **id** <B> ·

- Colección canónica de conjuntos de elementos
  - Se amplía la gramática añadiendo la regla <X> → <S>, siendo <S> el símbolo inicial.
  - Se calcula el conjunto I_0 = *clausura*(<X> → · <S>), que será el primer elemento de la colección.
  - Para cada conjunto de la colección (*C*) y cada símbolo terminal o no terminal (*A*) se calcula el conjunto *Ir_a(C,A)*, que se añade a la colección si no se había incluido previamente.
  - Se repite el paso anterior hasta que no se puedan añadir más conjuntos a la colección.
  - Si se representan los conjuntos de la colección como estados y las operaciones Ir_a() como transiciones, se obtiene un *Autómata reconocedor de prefijos viables*.

- Ejemplo de cálculo de una colección canónica de conjuntos de elementos



4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

4.6 Gestión de errores

4.7 Clasificación de gramáticas

- Algoritmo LR(0) de construcción de la tabla de análisis
  - Se obtiene la colección canónica de conjuntos de elementos.
  - Se obtiene el autómata reconocedor de prefijos viables.
  - Cada conjunto de la colección (I_i) es un estado del analizador (i).
  - Contenido de la sección *Ir_a* de la tabla de análisis:
    - Cada transición Ir_a(I_i,<A>) = I_j , siendo <A> un símbolo no terminal de la gramática, significa colocar el valor *j* en la celda Ir_a(i,<A>)

- Contenido de la sección *Acción* de la tabla de análisis
  - Cada transición $Ir\_a(I_i, a) = I_j$, siendo **a** un símbolo terminal, significa colocar el valor **dj** en la celda *Acción(i,a)*.
  - Para cada estado *i* que contenga el elemento (X → <S> ·) se coloca el valor **aceptar** en la celda *Acción(i,$)*.
  - Para los estados *i* que contengan un elemento con el punto al final de la regla k-ésima (<A> → β ·) colocar **rk** en todas las celdas del estado.
  - Las celdas que queden vacías representan errores sintácticos.

- Ejemplo

1: <S> → <B> <A> **end**
2: <A> → **begin** <C>
3: <C> → **codigo**
4: <B> → **tipo**
5: <B> → **id** <B>

Estado	**end**	**begin**	**codigo**	**tipo**	**id**	**$**	S	A	B	C
0				d3	d4		1		2	
1						aceptar				
2		d6						5		
3	r4	r4	r4	r4	r4	r4				
4				d3	d4				7	
5	d8									
6			d10							9
7	r5	r5	r5	r5	r5	r5				
8	r1	r1	r1	r1	r1	r1				
9	r2	r2	r2	r2	r2	r2				
10	r3	r3	r3	r3	r3	r3				

- El algoritmo LR(0) no requiere de la consideración de ningún token de preanálisis (de ahí el 0).
  - Cuando se alcanza un estado que sólo tiene acciones de reducción, se aplican éstas sin necesidad de estudiar el siguiente token de la cadena de entrada.
  - Cuando se alcanza un estado que no es de reducción, se lee el token de la cadena de entrada y se ejecuta la acción de desplazar correspondiente (o la de aceptar o la generación de un error).

4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

4.6 Gestión de errores

4.7 Clasificación de gramáticas

- Conflictos LR(0):
  - Los conflictos aparecen cuando una determinada celda de la sección Acción de la tabla puede rellenarse con varios valores.
  - Estos conflictos significan que la gramática no es LR(0). Puede que la gramática sea ambigua, o que sea LALR o LR(1).
- Ejemplo de conflicto LR(0):

1: <E> → <T> **plus** <E>
2: <E> → <T>
3: <T> → **number**

Estado	**number**	**plus**	**$**	E	T
0	d4			1	2
1			aceptar		
2	r2	**d3, r2**	r2		
3	d4			5	2
4	r3	r3	r3		
5	r1	r1	r1		

- En el ejemplo anterior, el estado que presenta el conflicto es el siguiente:
  - <E> → <T> · **plus** <E>
  - <E> → <T> ·
- El conflicto se debe a que en ese estado es posible reducir la regla 2 y también avanzar en el autómata consumiendo el token **plus**.
- El conflicto se podía haber evitado si consideramos que si después de <T> viene el token **plus** entonces no se debería reducir la regla 2 (sustituyendo <T> por <E>) ya que después de <E> no puede venir **plus**.
- Para evitar estos conflictos es necesario considerar los conjuntos siguientes de cada símbolo no terminal y tener en cuenta un token de preanálisis. Esto es la base del algoritmo SLR.

- Algoritmo SLR:
  - Es idéntico al algoritmo LR(0) salvo en la forma de rellenar las acciones de reducción.
  - Para los estados i que contengan un elemento con el punto al final de la regla k-ésima (<A> → b ·) colocar rk en todas las celdas de los tokens pertenecientes a Siguientes(<A>).
- Ejemplo:

1: <E> → <T> **plus** <E>
2: <E> → <T>
3: <T> → **number**

Estado	**number**	**plus**	**$**	E	T
0	d4			1	2
1			aceptar		
2		d3	r2		
3	d4			5	2
4		r3	r3		
5			r1		

4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

4.6 Gestión de errores

4.7 Clasificación de gramáticas

- Los errores sintácticos aparecen cuando en la sección Acción de la tabla no existe ninguna acción asociada al estado actual y al símbolo terminal de la entrada.
- El mensaje de error a generar es del tipo "*Encontrado token* **a**, *se esperaba uno de los siguientes:* **b**, **c**, **d**, …".
- El token encontrado (**a**) se refiere al token de la entrada y los tokens esperados (**b**, **c**, **d**, …) son aquellos que tengan acciones asociadas para el estado de la cima de la pila.

- La recuperación de errores en el análisis ascendente es muy compleja, ya que los estados pueden representar el grado de reconocimiento de varias reglas a la vez.
  - I_2 = { <S> → <A> · <B> **end**, <B> → · **begin** <C> }
  - Este estado tiene solo una acción asociada al token **begin**
- Para desarrollar una estrategia de recuperación de errores se añade una nueva regla con un token especial
  - <B> → · **error**
  - Cualquier token que no sea **begin** se considera un error
  - Cualquier token erróneo se elimina de la entrada y genera una reducción de esta regla.

- La herramienta YACC utiliza la siguiente técnica ante un error:
  - Desapila los estados hasta llegar a alguno que responda a una regla de error.
  - Desplaza la entrada y reduce la regla de error.
  - Desplaza la entrada hasta encontrar un token que no produzca error.

4.1 Introducción

4.2 Análisis sintáctico por desplazamiento y reducción

4.3 El autómata reconocedor de prefijos viables

4.4 Algoritmo LR(0)

4.5 Algoritmo SLR

4.6 Gestión de errores

4.7 Clasificación de gramáticas

- Gramáticas libres de contexto: se utilizan para el análisis sintáctico y pueden ser ambiguas. Es el conjunto más amplio.
- Gramáticas LR(k): son gramáticas no ambiguas que pueden analizarse por un método ascendente de orden k.
- Gramáticas LR(1): son gramáticas no ambiguas que pueden analizarse por un método ascendente lineal (k=1). Son un subconjunto de las LR(k).
- Gramáticas LALR(1): son un subconjunto de las LR(1) que pueden analizarse mediante el algoritmo LALR.
- Gramáticas SLR: son un subconjunto de las LALR que pueden analizarse mediante el algoritmo SLR.

- Gramáticas LL(k): son gramáticas no ambiguas que pueden analizarse por un método descendente con lookahead k. Son un subconjunto de LR(k).
- Gramáticas LL(1): son gramáticas no ambiguas que pueden analizarse por un método descendente con lookahead 1. Son un subconjunto de LL(k) y de LR(1).
- Es importante señalar que un lenguaje puede ser generado por muchas gramáticas de diferente tipo.