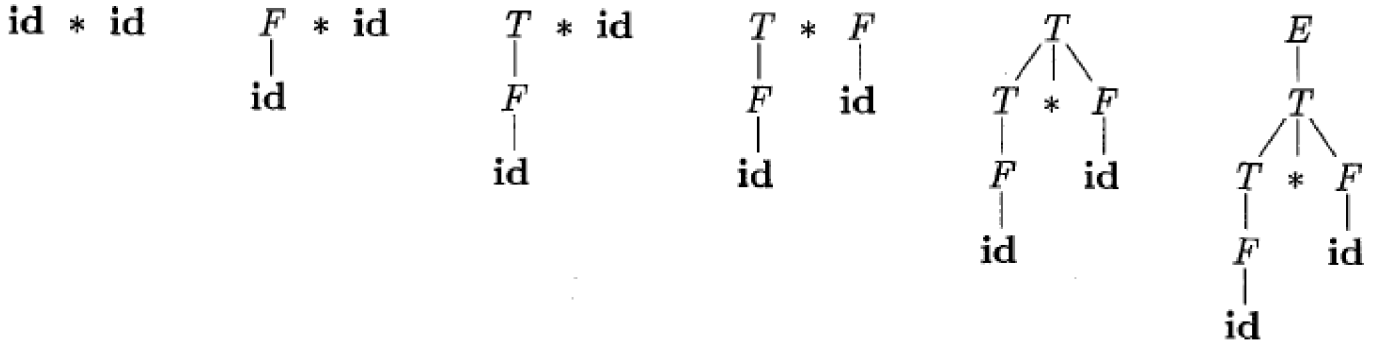
Compiladores:
Análisis Sintáctico

Pontificia Universidad Javeriana Cali
Ingenieria de Sistemas y Computación
Prof. Gloria Inés Alvarez V.

Análizador Sintáctico de abajo hacia arriba

- Es un proceso de Reducción, que construye el parser a partir de las hojas (símbolos de la cadena de entrada), y encuentra hacia arriba los nodos, hasta llegar a la raíz.
- En cada paso de la reducción, una subcadena de símbolos - que coincide con el lado derecho de una producción- es reemplazada con el no terminal del lado izquierdo de la producción.
- Encuentra la derivación más derecha

Ejemplo analizador de abajo hacia arriba [Aho p. 234]



Mando (handle) de una cadena

- "Handle" (mando o control) de una cadena es una subcadena que coincide con el lado derecho de una producción, y cuya reducción al no terminal de la izquierda representa un paso en la derivación por la derecha (en forma inversa) .

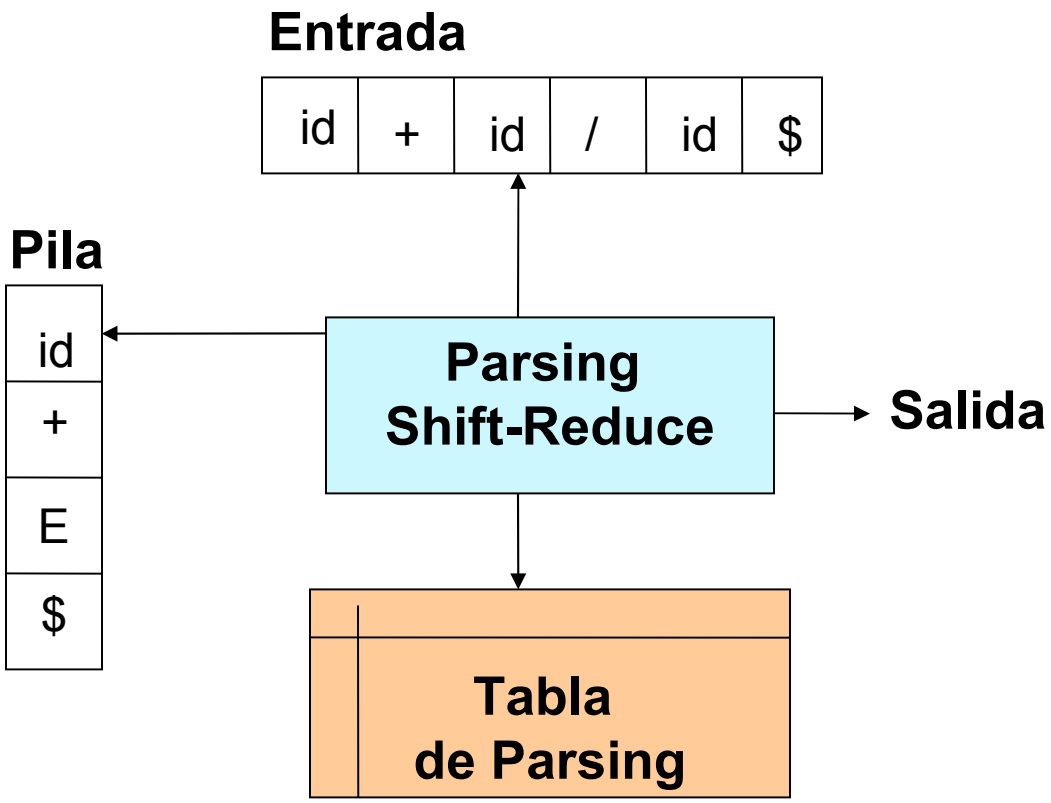
  Si $S \overset{*}{\Rightarrow} \alpha A w \overset{*}{\Rightarrow} \alpha\beta w$, entonces $A \longrightarrow \beta$ es un handle de $\alpha\beta w$ (la cadena w contiene simbolos terminales solamente)

- Poda de mandos: Obtiene de forma inversa la derivación por la derecha

Análizador Sintáctico Ascendente por desplazamiento/reducción

- Usa una pila y un puntero a la cadena de entrada
- Se basa en cuatro operaciones:
  - Desplazar: pone un símbolo de la entrada en el tope de la pila
  - Reducir: desapila algunos símbolos que son el cuerpo de una producción y los reemplaza por la cabeza
  - Aceptar
  - Detectar error

Análisis Sintáctico desplazamiento / reducción



El analizador sintáctico (parser), desplaza (shift) cero o más símbolos de la entrada al tope de la pila, hasta que un mando (handle) β esta en el tope de la pila. Entonces reduce β al no terminal del lado izquierdo de la regla de producción que aplica.

Análizador Sintáctico Ascendente por desplazamiento/reducción[Aho p237]

STACK	INPUT	ACTION
$	$\mathbf{id}_1 * \mathbf{id}_2$ $	shift
$ $\mathbf{id}_1$	$* \mathbf{id}_2$ $	reduce by $F \rightarrow \mathbf{id}$
$ F	$* \mathbf{id}_2$ $	reduce by $T \rightarrow F$
$ T	$* \mathbf{id}_2$ $	shift
$ $T *$	$\mathbf{id}_2$ $	shift
$ $T * \mathbf{id}_2$	$	reduce by $F \rightarrow \mathbf{id}$
$ $T * F$	$	reduce by $T \rightarrow T * F$
$ T	$	reduce by $E \rightarrow T$
$ E	$	accept

Conflictos durante el análisis desplazamiento/reducción

- Fallo desplazamiento/reducción
  - Ejemplo construcción if-then-else
- Fallo reducción/reducción
  - Ejemplo procedimiento-matriz

Cómo saber si desplazar o reducir?

- Para saberlo se debe llevar cuenta de qué parte de una producción ha sido ya reconocida en la cadena de entrada.
- Item LR(0): es una producción con un punto en algún sitio de la parte derecha
- Colección canónica LR(0)
- Autómata LR(0)

Items LR(0)

- Items:
  - Un ítem LR(0) de una gramática G, es una regla de producción de G, con un punto en alguna posición del lado derecho.
  - Por ejemplo, la regla $E \rightarrow E + T$, genera los ítems:
    - $E \rightarrow \bullet\ E + T$
    - $E \rightarrow E\ \bullet + T$
    - $E \rightarrow E + \bullet\ T$
    - $E \rightarrow E + T\ \bullet$
  - $A \rightarrow \epsilon$, genera el ítem: $A \rightarrow \bullet$

Items LR(0)

- Conjuntos de Items:
  - Un ítem no determina el estado de un parser
  - Por ejemplo, los ítems:
    - $E \rightarrow E \bullet + T$
    - $E \rightarrow E \bullet * T$
  - Para determinar el estado de un parser se consideran conjuntos de ítems
  - Los conjuntos de ítems serán los estados del autómata determinista LR(0).

Colección Canónica LR(0)

- Gramática Aumentada:
  - Si G es una gramática con símbolo inicial S, entonces, G' -la gramática G aumentada- es G con un nuevo símbolo de inicio S', tal que S' $\rightarrow$ S
  - Este nuevo símbolo (S') se crea para indicar al analizador que detenga el proceso y acepte la entrada (la cadena se acepta cuando se hace el reduce por S' $\rightarrow$ S)
- Clausura
- GoTo

Colección Canónica LR(0)

- Clausura:
  - Si I es un conjunto de ítems de una gramática G, clausura(I) es el conjunto de ítems:
    - Todo ítem de I se adiciona a clausura(I)
    - Si $A \rightarrow \alpha \bullet B\beta$ está en clausura(I), y $B \rightarrow \gamma$ es una regla de producción, entonces adicione $B \rightarrow \bullet\gamma$ (si no esta ya). Esta regla se aplica hasta que no puedan ser adicionados nuevos ítems.

Colección Canónica LR(0)

- Algoritmo clausura (I):

```
J = I
repita
  para cada A→ α•Bβ in J
    para cada B → γ tal que B → •γ no en J
      agregar B → •γ a J
hasta no más items para agregar a J
retorne J
```

Colección Canónica LR(0)

- Operación Goto:
  - Si I es un conjunto de ítems de una gramática G, y X es un símbolo de la gramática, goto(I, X) esta definido por la clausura de todos los ítems [$A \rightarrow \alpha X \bullet \beta$] tal que [$A \rightarrow \alpha \bullet X \beta$] esta en I.
  - Los goto son las transiciones del autómata determinista, goto (I, X) da la transición desde el estado I con el símbolo X.

Colección Canónica LR(0)

ítems (G')

```
C = clausura({S' → •S})
repita
 para cada conjunto de ítems I en C
  para cada símbolo X, de G
   si goto(I, X) != vacío y no esta en C
     agregue goto(I, X) a C
hasta que no haya mas conjuntos de ítems
  que puedan ser agregados a C
```

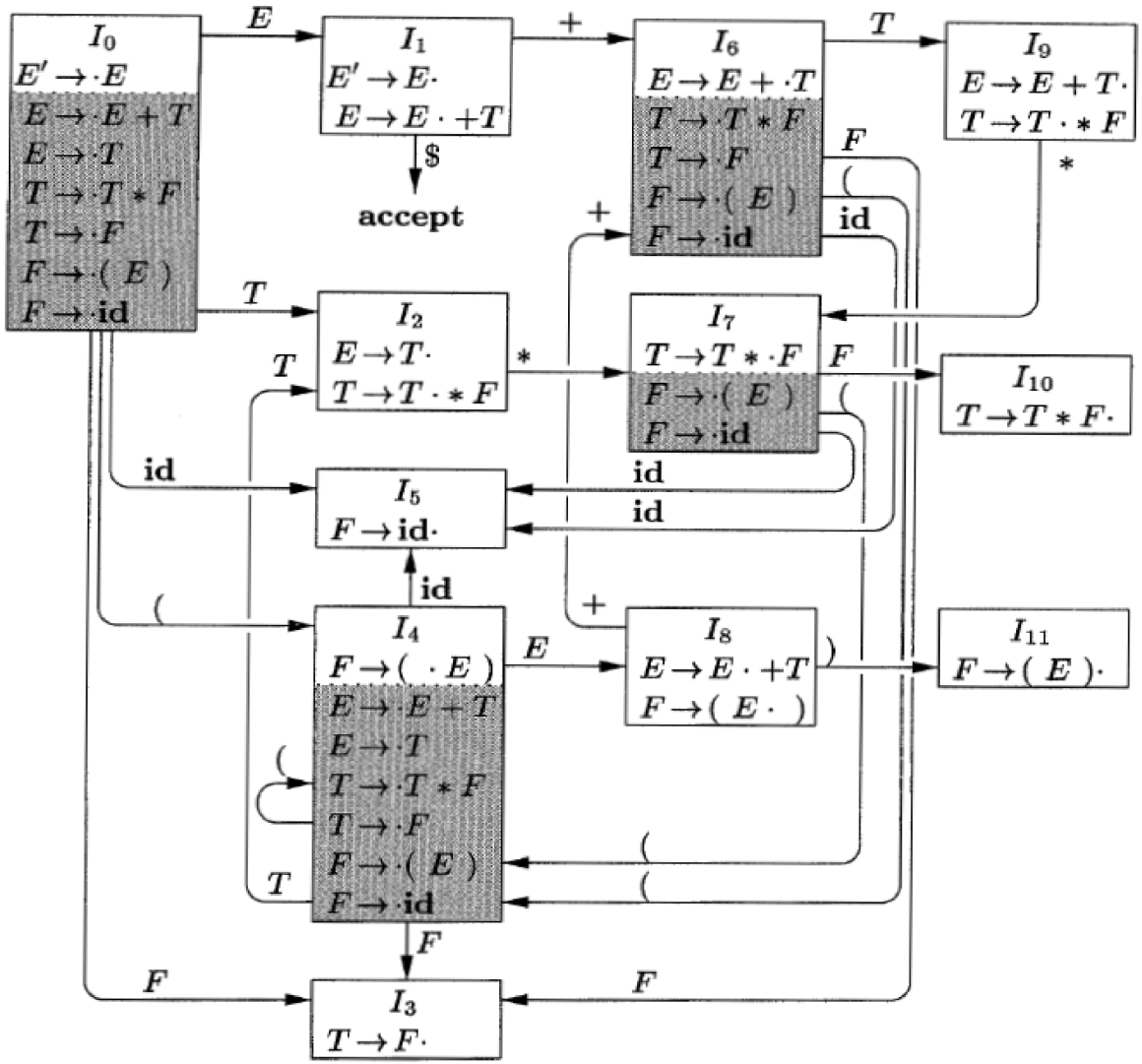
Autómata LR(0)

- Los estados son los elementos de la colección canónica LR(0)
- Las transiciones son los valores de la función GoTo
- Estado inicial: ({[S'$\rightarrow$ S]})
- Todos los estados son de aceptación
- Este autómata ayuda a decidir si se debe hacer shift o reduce cuando ambas opciones son posibles

Autómata LR(0)

- Si una cadena de símbolos de la gramática lleva el autómata del estado inicial a un estado 'j':
  - Se hace desplazamiento del siguiente símbolo en la entrada **a** si el estado 'j' tiene asociada alguna transición con ese símbolo
  - En caso contrario se hace reducción, los items del estado 'j' indican qué producción utilizar

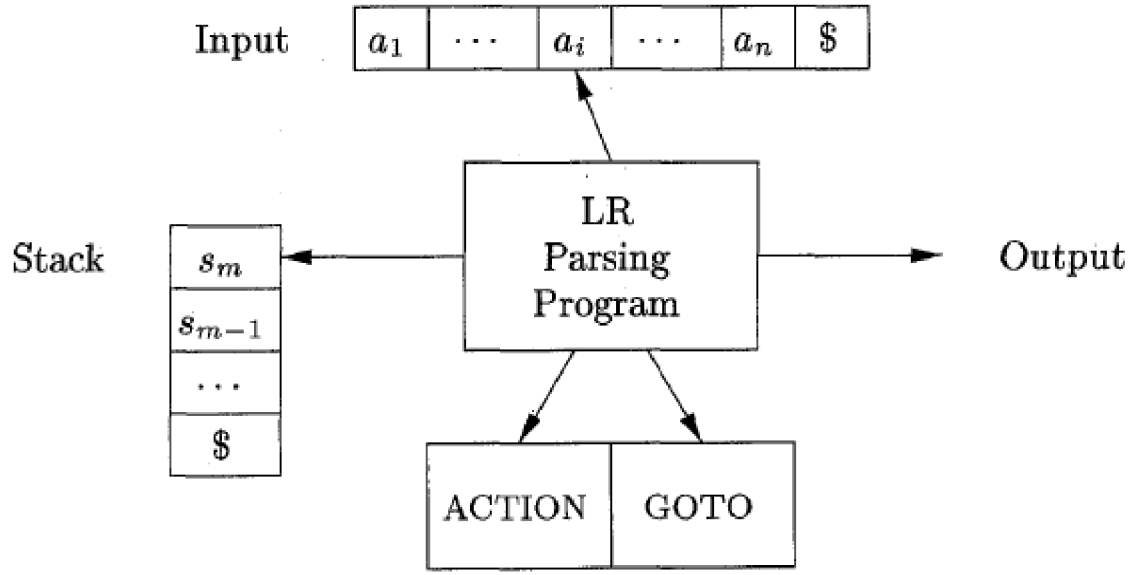
Autómata LR(0) [Aho p.244]



Análisis Sintáctico LR

- Pueden ser construidos para reconocer virtualmente todas las construcciones de los lenguajes de programación, para los que se puede escribir una gramática libre de contexto.
- Es el método más general de parser desplazamiento/reducción sin backtracking.
- Las gramáticas que pueden ser analizadas con parser predictivo (top-down) son un subconjunto de las gramáticas que pueden ser analizadas con parser LR.
- Detecta errores sintácticos tan pronto como es posible hacerlo recorriendo izquierda-a-derecha de la entrada

Algoritmo de análisis sintáctico LR [Aho p.248]



Algoritmo LR [Aho p. 251]

METHOD: Initially, the parser has s_0 on its stack, where s_0 is the initial state, and $w\$$ in the input buffer. The parser then executes the program in Fig. 4.36. □

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Tablas para algoritmo LR

- Simple LR (SLR)
- Canonical LR
- Look Ahead LR (LALR)

Ejemplo
Tabla SLR(0) [Aho p. 252]

STATE	ACTION						GOTO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Construcción Tabla SLR(0) [Aho p.253-254]

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.

2. State i is constructed from I_i. The parsing actions for state i are determined as follows:

   (a) If $[A \to \alpha \cdot a\beta]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set $\text{ACTION}[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot]$ is in I_i, then set $\text{ACTION}[i, a]$ to "reduce $A \to \alpha$" for all a in $\text{FOLLOW}(A)$; here A may not be S'.

   (c) If $[S' \to S \cdot]$ is in I_i, then set $\text{ACTION}[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.

Construcción Tabla SLR(0)

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.
4. All entries not defined by rules (2) and (3) are made "error."
5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S]$.

Gramáticas LR

- Una gramática para la cual se puede construir una tabla de parsing LR.
- Significa que un parser shift-reduce, que lee la entrada de izquierda a derecha, sea capaz de reconocer los mandos, cuando aparecen en el tope de la pila.
- El parser toma la decisión de acuerdo con el estado que está en el tope de la pila, y con los siguientes *k* símbolos de la cadena de entrada $\Rightarrow$ gramática *LR(k)* (reconoce la ocurrencia del lado derecho de una producción, teniendo lo que se ha derivado y k símbolos hacia adelante)
- Las gramáticas LR pueden describir mas lenguajes que las gramáticas LL.

FIN

Parsing LR: Ejemplo:

- $E \rightarrow E + T$
- $E \rightarrow T$
- $T \rightarrow T * id$
- $T \rightarrow id$

S_n : shift al estado n

R_m: reduce con la regla (m)

Acc: accept

Espacio en blanco: error.

	Action				Goto	
	Id	**+**	*****	**$**	**E**	**T**
I_0	S_3				1	2
I_1		S_4		Acc		
I_2		r_2	S_5	r_2		
I_3		r_4	r_4	r_4		
I_4	S_3					6
I_5	S_7					
I_6		r_1	S_5	r_1		
I_7		r_3	r_3	r_3		