Compiladores
Facultad de Ciencias UNAM
Análisis Sintáctico: *parser* ***LR(0)*** [*]

Lourdes Del Carmen González Huesca [**]

24 de marzo de 2024

Gramática de expresiones aritméticas sencilla

El siguiente ejemplo muestra la construcción de la tabla de parsing LR(0) para la gramática de expresiones aritméticas sencilla y se mostrará el análisis de una cadena.
La gramática a usar puede ser una que incluya recursión izquierda ya que los parsers **LR** admiten este tipo de gramáticas.

Para comenzar, la gramática será extendida con el símbolo E' para denotar al nuevo símbolo inicial. Cada producción está numerada, la producción inicial (0) en la gramática extendida es $E' \to E$:

$$\begin{array}{lrcl} (0) & E' & \to & E \\ (1) & E & \to & E + T \\ (2) & E & \to & T \\ (3) & T & \to & T * F \\ (4) & T & \to & F \\ (5) & F & \to & (E) \\ (6) & F & \to & n \end{array}$$

Recordemos que la tabla se construye con ayuda de un autómata finito para decidir las acciones el autómata de pila que es el parser. Los estados de la máquina finita serán los conjuntos canónicos de items.

- Se iniciará el cálculo de los estados y sus transiciones con el item $E' \to \bullet E$. El estado I_0 será la cerradura de este item.

> Sea I un conjunto de items, la cerradura de I o $\textsc{Closure}(I)$ son los items tales que
>
> 1. es elemento de I
> 2. Si $A \to \alpha \bullet B\beta$ está en $\textsc{Closure}(I)$ y $B \to \gamma$ es una producción de la gramática, entonces agregar $B \to \bullet\gamma$ a $\textsc{Closure}(I)$

$$\textsc{Closure}(\{E' \to \bullet E\}) = \{E' \to \bullet E,\ E \to \bullet E{+}T,\ E \to \bullet T,\ T \to \bullet T{*}F,\ T \to \bullet F,\ F \to \bullet(E),\ F \to \bullet n\}$$

[*]Este ejemplo (incluyendo las imágenes) está tomado del capítulo 4 del libro "Compilers, Principles, Techniques and Tools", Aho A. et al.

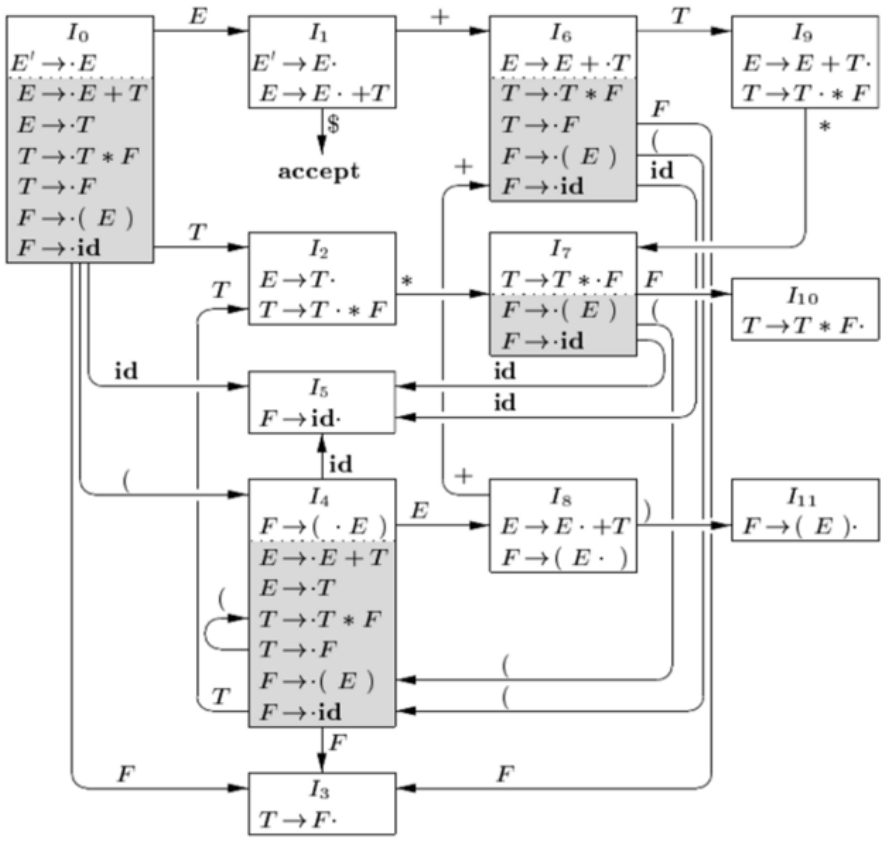
[**]Material revisado por el servicio social de Apoyo a la Docencia y Asesoría Académica en la Facultad de Ciencias de la UNAM con clave 2023-12/12-292, de Diana Laura Nicolás Pavia.

A continuación se calcularán las funciones GoTo de este estado para generar los items cuya cerradura generarán los siguientes estados del autómata finito.

> Dado un conjunto de items I y un símbolo cualquiera de la gramática X, la función GoTo del estado I mediante X es la cerradura de todos los items $A \to \alpha X \bullet \beta$ tal que $A \to \alpha \bullet X\beta$ está en I.

$\Sigma \cup V$	GoTo I_0	nuevo estado
E	Closure($\{E' \to E\bullet,\ E \to E \bullet +T\}$)	I_1
T	Closure($\{E \to T\bullet,\ T \to T \bullet *F\}$)	I_2
F	Closure($\{T \to F\bullet\}$)	I_3
(	Closure($\{F \to (\bullet E)\}$)	I_4
n	Closure($\{F \to n\bullet\}$)	I_5
$\{\ ), +, *, \#\ \}$	$\varnothing$	

- El proceso anterior se repite para cada estado nuevo generado hasta completar el autómata. El autómata para esta gramática es [1]:



La transición a la aceptación es la que se obtiene al procesar el símbolo de fin de cadena y donde el item $E' \to E\bullet$ pertenece al estado en cuestión.

- Una vez construido el autómata se puede usar para completar la tabla de parsing considerando lo siguiente:

[1] En la figura aparece el símbolo $ como fin de cadena.

Acciones la parte de acciones está determinada por las transiciones del autómata y un símbolo cualquiera de la gramática (etiqueta de una transición):

- *shift* $(s\, i)$ almacenar el símbolo terminal y desplazarse al estado i
- *reduce* $(r\, j)$ reducir con la producción j y no cambia de estado
- *accept*
- *error*

GoTo la parte de transiciones determinada por un símbolo no-terminal indicando el nuevo estado al que se desplaza.

La tabla correspondiente es la siguiente [2]:

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$

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			

Veamos el análisis de la cadena **id** $\star$ **id** siguiendo el algoritmo:

```
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→v )
            {
              pop v 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→v;
            } else if ( ACTION[s; a] = accept ) break;
                   /* parsing is done */
            else call error-recovery routine;
}
```

LINE	STACK	SYMBOLS	INPUT	ACTION
(1)	0	$	**id** * **id** $	shift to 5
(2)	0 5	$ **id**	* **id** $	reduce by $F \to \mathbf{id}$
(3)	0 3	$ F	* **id** $	reduce by $T \to F$
(4)	0 2	$ T	* **id** $	shift to 7
(5)	0 2 7	$ T *	**id** $	shift to 5
(6)	0 2 7 5	$ T * **id**	$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$ T * F	$	reduce by $T \to T * F$
(8)	0 2	$ T	$	reduce by $E \to T$
(9)	0 1	$ E	$	accept

[2] En la figura aparece el símbolo $ como fin de cadena.