GRUPO 63

Procesador de Lenguajes: Analizador Sintáctico

Procesadores de Lenguajes

Tabla de contenido

Ignacio García Fernández
José Ruiz Esteban
Eduardo Gil Alba

18-11-2024

1. Propósito

En este documento se refleja el procedimiento de diseño para un analizador sintáctico del lenguaje Javascript JS--, para un conjunto de sentencias, de dicho lenguaje. Se muestran más adelante. Se ha dejado planteado su diseño, pero no hemos podido implementarlo en esta entrega.

Conjunto de sentencias

- La estructura general de un programa compuesto por funciones y declaraciones.
- Definición de funciones. (**function**, **return**)
- Tipos enteros, lógicos, cadenas y vacío. (**ent**, **bool**, **cad**, **void**)
- Variables y su declaración. (**var**, **int**, **string**, **boolean**)
- Constantes enteras y cadenas de caracteres.
- Sentencias:
    - Asignación:
        - Asignación con y lógico (**&=**)
    - Condicionales:
        - Sentencia condicional compuesta (**if**, **if-else**)
    - Llamada a funciones y retorno
- Sentencias de entrada/salida por terminal. (**output**, **input**)
- Expresiones.
    - Aritméticas:
        - Suma (**+**)
    - Relacionales:
        - Igualdad (**==**)
    - Lógicas:
        - Y lógico (**&&**)
- Comentarios.
    - Comentarios de Bloque (/* */)
- Cadenas:
    - Con comillas simple (‘ ‘)

2. Analizador Sintáctico

El analizador sintáctico empleará los tokens que recibe del analizador léxico y evaluará si la sentencia es válida para el lenguaje generado por la gramática.

2.1. Gramática

La gramática contiene la sintaxis del lenguaje. En otras palabras, el conjunto de sentencias válidas para nuestro lenguaje. La gramática debe ser de tipo 2, independiente del contexto, lo más general posible, para cubrir todos los posibles casos de uso y no debe ser ambigua, es decir, no puede dar lugar a dos o más arboles sintácticos diferentes.

Gramática de An. St. Ascendente

Estructura del programa

```
1 : P → BP
2 : P → FP
3 : P → eof
```

Sentencias Compuestas

```
4 : B → var T id ;
5 : B → if ( E ) { C } I
6 : B → if ( E ) S
7 : B → S
8: F → function F1 F2 F3 { C }
9 : F1 → H
10 : F2 → id
11 : F3 → ( A )
12 : C → BC
13 : C → λ
14 : H → T
15 : H → void
16 : A → T id K
17 : A → void
18 : K → , T id K
19 : K → λ
20 : I → else { C }
21 : I → λ
22 : T → int
23 : T → boolean
24 : T → string
```

Sentencias Simples

```
25 : S → id = E ;
26 : S → id &= E ;
27 : S → id ( L ) ;
28 : S → output E ;
29 : S → input id ;
30 : S → return X ;
31 : L → EQ
32 : L → λ
33 : Q → , EQ
34 : Q → λ
35 : X → E
36 : X → λ
```

Expresiones

```
37 : E → E && R
38 : E → R
39 : R → R == U
40 : R → U
41 : U → U + V
42 : U → V
43 : V → id
44 : V → ( E )
45 : V → id ( L )
46 : V → ent
47 : V → cad
48 : V → bool
```

2.2. Tabla de Analizador Sintáctico LR(1)

E	ACCION																											
	id	+	==	&&	&=	=	(	)	{	}	;	,	function	var	return	if	else	int	boolean	string	void	ent	cad	bool	output	input	eof	$
0	d9												d8	d5	d12	d6									d10	d11	d4	
1																												a
2	d9												d8	d5	d12	d6									d10	d11	d4	
3	d9												d8	d5	d12	d6									d10	d11	d4	
4																												r3
5																		d16	d17	d18								
6							d19																					
7	r7									r7			r7	r7	r7	r7									r7	r7	r7	
8																		d16	d17	d18	d23							
9					d25	d24	d26																					
10	d32						d31															d33	d34	d35				
11	d36																											
12	d32						d31				r36											d33	d34	d35				
13																												r1
14																												r2
15	d39																											
16	r22																											
17	r23																											
18	r24																											

E	ACCION																										
	id	+	==	&&	&=	=	(	)	{	}	;	,	function	var	return	if	else	int	boolean	string	void	ent	cad	bool	output	input	eof	$
19	d32						d31															d33	d34	d35				
20	d42																											
21	r9																											
22	r14																											
23	r15																											
24	d32						d31															d33	d34	d35				
25	d32						d31															d33	d34	d35				
26	d32						d31															d33	d34	d35				
27				d48							d47																	
28			d49	r38				r38			r38	r38																
29		d50	r40	r40				r40			r40	r40																
30		r42	r42	r42				r42			r42	r42																
31	d32						d31															d33	d34	d35				
32		r43	r43	r43			d52	r43			r43	r43																
33		r45	r45	r45				r45			r45	r45																
34		r47	r47	r47				r47			r47	r47																
35		r48	r48	r48				r48			r48	r48																
36											d53																	
37											d54																	

E	ACCION																											
	id	+	==	&&	&=	=	(	)	{	}	;	,	function	var	return	if	else	int	boolean	string	void	ent	cad	bool	output	input	eof	$
38				d48							r35																	
39											d55																	
40				d48				d56																				
41							d58																					
42	r10																											
43				d48							d59																	
44				d48							d60																	
45								d61																				
46				d48				r34				d63																
47	r28									r28			r28	r28	r28	r28									r28	r28	r28	
48	d32						d31															d33	d34	d35				
49	d32						d31															d33	d34	d35				
50	d32						d31															d33	d34	d35				
51				d48				d67																				
52	d32						d31	r32														d33	d34	d35				
53	r29									r29			r29	r29	r29	r29									r29	r29	r29	
54	r30									r30			r30	r30	r30	r30									r30	r30	r30	
55	r4									r4			r4	r4	r4	r4									r4	r4	r4	
56	d9								d69						d12										d10	d11		

E	ACCION																										
	id	+	==	&&	&=	=	(	)	{	}	;	,	function	var	return	if	else	int	boolean	string	void	ent	cad	bool	output	input	eof	$
57									d71																			
58																		d16	d17	d18	d74							
59	r25									r25			r25	r25	r25	r25									r25	r25	r25	
60	r26									r26			r26	r26	r26	r26									r26	r26	r26	
61											d75																	
62								r31																				
63	d32						d31															d33	d34	d35				
64			d49	r37				r37			r37	r37																
65			r39	r39				r39			r39	r39																
66		r41	r41	r41				r41			r41	r41																
67		r44	r44	r44				r44			r44	r44																
68								d77																				
69	d9									r13				d5	d12	d6									d10	d11		
70	r6									r6			r6	r6	r6	r6									r6	r6	r6	
71	d9									r13				d5	d12	d6									d10	d11		
72								d81																				
73	d82																											
74								r17																				
75	r27									r27			r27	r27	r27	r27									r27	r27	r27	

E	ACCION																											
	id	+	==	&&	&=	=	(	)	{	}	;	,	function	var	return	if	else	int	boolean	string	void	ent	cad	bool	output	input	eof	$
76				d48				r34				d63																
77		r46	r46	r46				r46			r46	r46																
78										d84																		
79	d9									r13				d5	d12	d6									d10	d11		
80										d86																		
81									r11																			
82								r19				d88																
83								r33																				
84	r21									r21			r21	r21	r21	r21	d90								r21	r21	r21	
85										r12																		
86	r8												r8	r8	r8	r8									r8	r8	r8	
87								r16																				
88																		d16	d17	d18								
89	r5									r5			r5	r5	r5	r5									r5	r5	r5	
90									d92																			
91	d93																											
92	d9									r13				d5	d12	d6									d10	d11		
93								r19				d88																
94										d96																		

E	ACCION																											
	id	+	==	&&	&=	=	(	)	{	}	;	,	function	var	return	if	else	int	boolean	string	void	ent	cad	bool	output	input	eof	$
95								r18																				
96	r20									r20			r20	r20	r20	r20									r20	r20	r20	

Nuestro analizador sintáctico, es un analizador sintáctico ascendente, es decir, el árbol generado por este analizador se construye desde las hojas. En concreto, necesitará un token, en cada iteración. Para su implementación, hemos construido una tabla que determina qué acción realizará nuestro analizador en un determinado estado, compuesta por los tokens en cada columna y los estados en las filas. Las acciones definidas pueden ser reducir por la Regla i (r i), desplazar el estado m (d m), aceptar o error (las celdas no definidas de la tabla, denotadas con sombreado). De tal forma que, las acciones de reducción generarán un “parse”, una secuencia de reglas, que determina la construcción del árbol sintáctico, sin necesidad de una estructura para ello. Tendremos una pila, en la que iremos introduciendo, pares de símbolos y estados. En cada iteración, el estado de la cima de la pila y el token proporcionado por el léxico determina la siguiente acción.

Análisis de Conflictos

Existe un conflicto, si en un estado para un determinado token, existen dos o más acciones. No se podría determinar cuál de las acciones se debe ejecutar. Los conflictos que se pudieran producir serían de tipo reducción-reducción o reducción-desplazamiento. Dicho de otro modo, en el primer caso, que hubiese dos o más acciones de reducción para un mismo token y en el segundo, una o más acciones de reducción y una o más acciones de desplazar.

En nuestro caso, se pudieran haber presentado conflictos de reducción-desplazamiento, en 16 estados. Los estados 12, 26, 28, 29, 38, 46, 52, 65, 69, 71, 76, 79, 82, 84, 92 y 93. Se observa que, cada columna, token, en ese estado, fila, tiene una sola acción. Por lo tanto, está gramática es válida y la tabla está lista para su implementación.

ANEXO

Casos de prueba

No hemos conseguido implementarlo en esta entrega.

Tabla GOTO

E	GOTO																			
	P	B	F	T	E	C	I	S	F1	F2	F3	H	A	K	L	X	Q	R	U	V
0	1	2	3					7												
1																				
2	13	2	3					7												
3	14	2	3					7												
4																				
5				15																
6																				
7																				
8				22					20			21								
9																				
10					27													28	29	30
11																				
12					38											37		28	29	30
13																				
14																				
15																				
16																				
17																				
18																				
19					40													28	29	30
20										41										
21																				
22																				
23																				
24					43													28	29	30
25					44													28	29	30
26					46										45			28	29	30
27																				
28																				
29																				

E	GOTO																			
	P	B	F	T	E	C	I	S	F1	F2	F3	H	A	K	L	X	Q	R	U	V
30																				
31					51													28	29	30
32																				
33																				
34																				
35																				
36																				
37																				
38																				
39																				
40																				
41											57									
42																				
43																				
44																				
45																				
46																	62			
47																				
48																		64	29	30
49																			65	30
50																				66
51																				
52					46										68			28	29	30
53																				
54																				
55																				
56								70												
57																				
58				73									72							
59																				
60																				
61																				
62																				
63					76													28	29	30

E	GOTO																			
	P	B	F	T	E	C	I	S	F1	F2	F3	H	A	K	L	X	Q	R	U	V
63					76													28	29	30
64																				
65																				
66																				
67																				
68																				
69		79				78		7												
70																				
71		79				80		7												
72																				
73																				
74																				
75																				
76																	83			
77																				
78																				
79		79				85		7												
80																				
81																				
82														87						
83																				
84							89													
85																				
86																				
87																				
88				91																
89																				
90																				
91																				
92		79				94		7												
93														95						
94																				
95																				
96																				