



E.T.S.I. Informáticos
Universidad Politécnica de Madrid



Procesadores de Lenguajes

Práctica

Analizador Sintáctico

José Luis Fuertes

noviembre de 2020

1

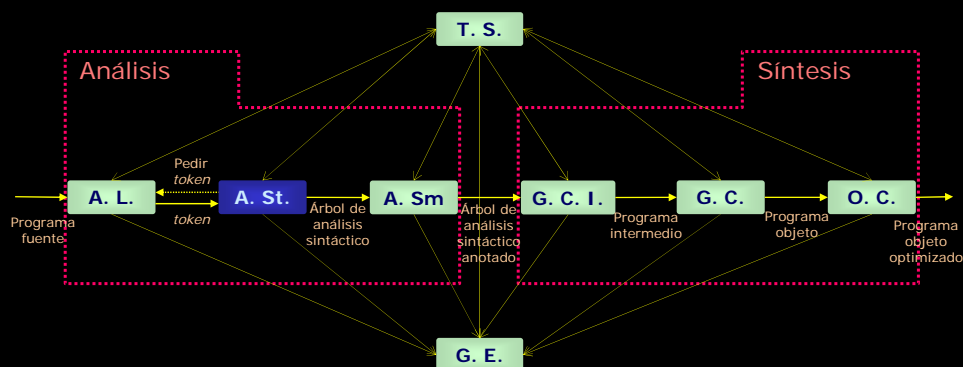
Contenido

1. Elementos del lenguaje JavaScript-PDL
2. ¿Cómo construir una gramática de un lenguaje?
3. Construcción de la gramática para el Analizador Sintáctico de JavaScript-PDL
4. ¿Y una vez que tengo la gramática?
5. Implementación del Analizador Sintáctico

2

2

Compilador



3

Análisis Sintáctico

3

Elementos del Lenguaje JavaScript-PDL

- Es fundamental conocer bien el lenguaje
 - ♦ <https://dlsiisv.fi.upm.es/procesadores/IntroJavaScript.html>
- JavaScript-PDL tiene varios elementos
 - ♦ Declaraciones
 - de variables
 - de funciones
 - ♦ Sentencias
 - simples
 - compuestas
 - ♦ Expresiones
 - se utilizan en las sentencias

4

Práctica-ASt

4

¿Cómo construir una gramática de un lenguaje?

- Recordar el tipo de gramática para un Analizador Sintáctico
 - ♦ Tipo 2 = Independiente del contexto
- Escribir las reglas para cada parte del lenguaje teniendo en cuenta la estructura sintáctica del lenguaje y todas las posibilidades
- Conectar entre sí los distintos grupos de reglas para formar la gramática completa
- La gramática no puede ser ambigua

5

Práctica-ASt

5

¿Cómo construir una gramática de un lenguaje?

- Declaraciones
 - ♦ Permiten declarar nombres (variables, funciones, tipos, clases, etc.)
- Sentencias
 - ♦ Son las construcciones de un lenguaje que ejecutan una acción
 - simple
 - compuesta (bucles, condicionales, etc.)
- Expresiones
 - ♦ Son las construcciones de un lenguaje que permiten calcular un valor
 - ♦ Suelen estar formadas por
 - operandos
 - operadores
 - paréntesis
 - ♦ Pueden ser de cualquiera de los tipos del lenguaje
 - ♦ Usar el tipo de gramática utilizado en clase

6

Práctica-ASt

6

Construcción de la gramática de JavaScript-PDL

• Expresiones

- ♦ Es fundamental tener en cuenta la precedencia y asociatividad de los operadores

■ <https://dlsiisv.fi.upm.es/procesadores/IntroJavaScript.html#Operadores>

■ Tabla:

- arriba: menos precedencia
- abajo: más precedencia

■ Precedencia

- $a + b * c = a + (b * c)$

■ Asociatividad

- $a * b / c = (a * b) / c$

Operadores	Significado	Asociatividad
	O lógico	Izquierda a derecha
&&	Y lógico	Izquierda a derecha
== !=	Igual Distinto	Izquierda a derecha
> >= < <=	Mayor Mayor o igual Menor Menor o igual	Izquierda a derecha
+ -	Suma Resta	Izquierda a derecha
* / %	Producto División Módulo	Izquierda a derecha
! ++ -- + -	Negación lógica Autoincremento Autodecremento Más unario Menos unario	Derecha a izquierda

Práctica-ASt

7

7

Construcción de la gramática de JavaScript-PDL

• Expresiones

- ♦ $E \rightarrow E \&\& R \mid R$
- ♦ $R \rightarrow R > U \mid U$
- ♦ $U \rightarrow U + V \mid V$
- ♦ $V \rightarrow id \mid (E) \mid id (L) \mid ent \mid cad$

Precedencia

Operadores	Significado	Asociatividad
	O lógico	Izquierda a derecha
&&	Y lógico	Izquierda a derecha
== !=	Igual Distinto	Izquierda a derecha
> >= < <=	Mayor Mayor o igual Menor Menor o igual	Izquierda a derecha
+ -	Suma Resta	Izquierda a derecha
* / %	Producto División Módulo	Izquierda a derecha
! ++ -- + -	Negación lógica Autoincremento Autodecremento Más unario Menos unario	Derecha a izquierda

Práctica-ASt

8

8

Construcción de la gramática de JavaScript-PDL

- Sentencias simples

- ♦ $S \rightarrow \text{id} = E ;$

- ♦ $S \rightarrow \text{id} (L) ;$

- ♦ $S \rightarrow \text{alert} (E) ;$

- ♦ $S \rightarrow \text{input} (\text{id}) ;$

- ♦ $S \rightarrow \text{return } X ;$

- ♦ $L \rightarrow E Q \mid \lambda$

- ♦ $Q \rightarrow , E Q \mid \lambda$

- ♦ $X \rightarrow E \mid \lambda$

9

Práctica-ASt

9

Construcción de la gramática de JavaScript-PDL

- Sentencias compuestas y declaración de variables

- ♦ $B \rightarrow \text{if} (E) S$

- ♦ $B \rightarrow \text{let } T \text{ id} ;$

- ♦ $B \rightarrow S$

- ♦ $T \rightarrow \text{number} \mid \text{boolean} \mid \text{string}$

10

Práctica-ASt

10

Construcción de la gramática de JavaScript-PDL

- Declaración de funciones

$$\blacklozenge F \rightarrow \text{function } H \text{ id } (A) \{ C \}$$

$$\blacklozenge H \rightarrow T \mid \lambda$$

$$\blacklozenge A \rightarrow T \text{ id } K \mid \lambda$$

$$\blacklozenge K \rightarrow , T \text{ id } K \mid \lambda$$

$$\blacklozenge C \rightarrow B C \mid \lambda$$

- Otra posibilidad (*recomendable para A. St. Ascendente*)

$$\blacksquare F \rightarrow F1 F2 F3$$

$$\blacksquare F1 \rightarrow \text{function } H \text{ id}$$

$$\blacksquare F2 \rightarrow (A)$$

$$\blacksquare F3 \rightarrow \{ C \}$$

11

Práctica-ASt

11

Construcción de la gramática de JavaScript-PDL

- Estructura de un programa = Axioma

$$\blacklozenge P \rightarrow B P$$

$$\blacklozenge P \rightarrow F P$$

$$\blacklozenge P \rightarrow \lambda$$

12

Práctica-ASt

12

¿Y una vez que tengo la gramática?

- La gramática debe ser adecuada para el Analizador Sintáctico
 - ♦ Puede ser necesario transformarla
 - ♦ A. St. Descendente
 - Gramática LL(1)
 - Factorizar
 - Eliminar recursividad por la izquierda
 - Verificar Condición LL(1)
 - ♦ A. St. Ascendente
 - Gramática LR(1)
 - Analizador sin conflictos
 - reducción / desplazamiento
 - reducción / reducción

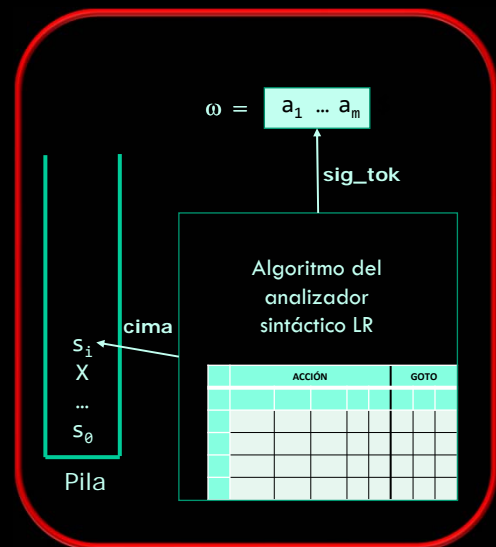
13

Práctica-ASt

13

Implementación del Analizador Sintáctico

- Analizador Sintáctico Ascendente LR
 1. Diseñar el Autómata reconocedor de prefijos viables
 - Comprobar que no hay conflictos
 - Obtener las tablas
 - Acción
 - Goto
 2. Programar las tablas en una estructura de datos
 3. Programar el algoritmo de Análisis Ascendente



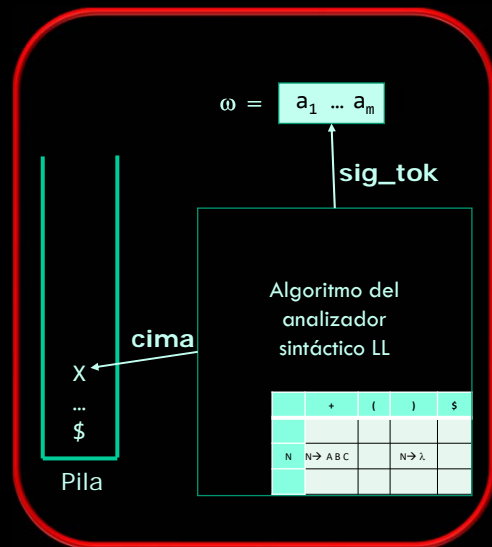
14

Práctica-ASt

14

Implementación del Analizador Sintáctico

- Analizador Sintáctico Descendente con Tabla
 1. Diseñar la tabla del Analizador
 2. Programar la tablas en una estructura de datos
 3. Programar el algoritmo de Análisis Descendente



15

Práctica-ASt

15

Implementación del Analizador Sintáctico

- Analizador Sintáctico Descendente Recursivo
 1. Diseñar en pseudo-código los procedimientos de cada símbolo No Terminal
 2. Implementar dichos procedimientos

```

...
7. F → ( E )
8. F → id
...

Function F ()
{
  if sig_tok == <(>) then
  {
    equipara (<(>)
    E;
    equipara (<)>)
  }
  else
  {
    equipara (<id>)
  }
}

```

16

Práctica-ASt

16

Implementación del Analizador Sintáctico

- Ficheros de salida
 - ♦ Formatos: <https://dlsiisv.fi.upm.es/procesadores/Documentacion.html>
 - ♦ Necesarios para
 - depurar vuestro analizador
 - probar vuestro analizador en VASt
 - probar vuestro analizador en Draco
 - corregir la práctica en la entrega final
 - ♦ Fichero: Gramática del Analizador Sintáctico
 - ♦ Fichero: Parse
- Herramienta VASt
 - ♦ Visualizador de Árboles Sintácticos
 - ♦ <https://dlsiisv.fi.upm.es/procesadores/Herramientas.html>
 - ♦ A partir de la gramática y del *parse* generado, construye el árbol correspondiente al programa fuente

