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

Procesadores de Lenguajes

Práctica

Analizador Sintáctico

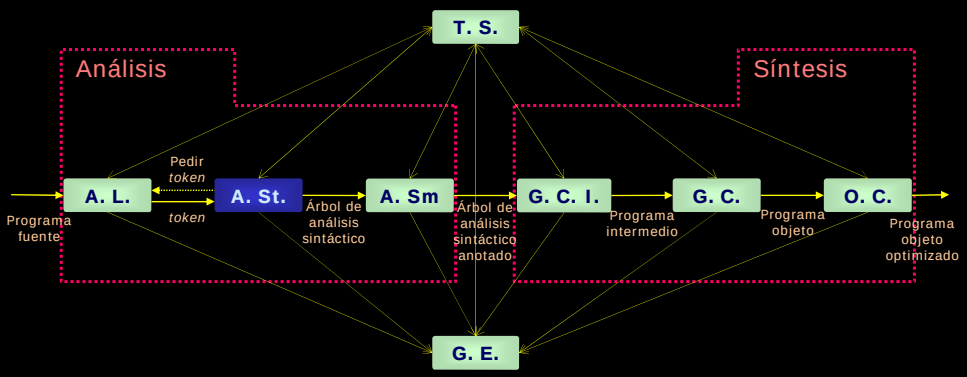
José Luis Fuertes

octubre de 2022

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

Compilador



Elementos del Lenguaje JavaScript-PDL

- Es fundamental conocer bien el lenguaje
  - https://dlsiis.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

¿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

¿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

Construcción de la gramática de JavaScript-PdL

- Expresiones
  - Es fundamental tener en cuenta la precedencia y asociatividad de los operadores
    - https://dlsiis.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

Construcción de la gramática de JavaScript-PdL

- Expresiones
  - E → E && R | R
  - R → R > U | U
  - U → U + V | V
  - V → id | (E) | id (L) | ent | 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

Construcción de la gramática de JavaScript-PdL

- Sentencias simples
  - S → id = E ;
  - S → id (L) ;
  - S → print E ;
  - S → input id ;
  - S → return X ;
  - L → E Q | λ
  - Q → , E Q | λ
  - X → E | λ

Construcción de la gramática de JavaScript-PdL

- Sentencias compuestas y declaración de variables
  - B → if (E) S
  - B → let id T ;
  - B → S
  - B → *while* | *do-while* | *for* | *switch* | *if-else*
  - T → int | boolean | string

Construcción de la gramática de JavaScript-PdL

- Declaración de funciones
  - F → function id H (A) { C }
  - H → T | λ
  - A → T id K | λ
  - K → , T id K | λ
  - C → B C | λ
  - Otra posibilidad (*recomendable para A. St. Ascendente*)
    - F → F1 F2 F3
    - F1 → function id H
    - F2 → (A)
    - F3 → { C }

Construcción de la gramática de JavaScript-PdL

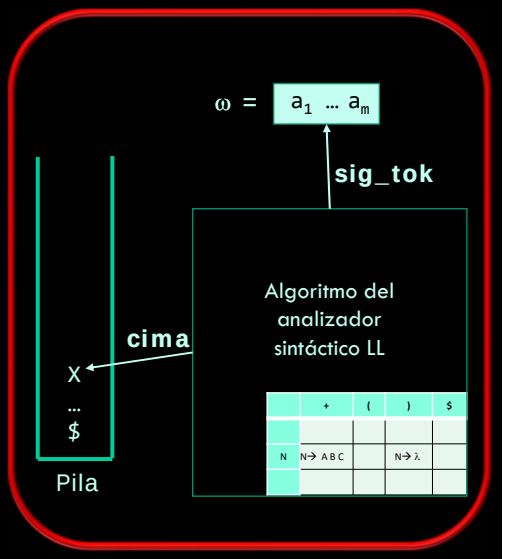
- Estructura de un programa = Axioma
  - P → B P
  - P → F P
  - P → λ

¿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

Implementación del Analizador Sintáctico

- Análizador 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



Implementación del Analizador Sintáctico

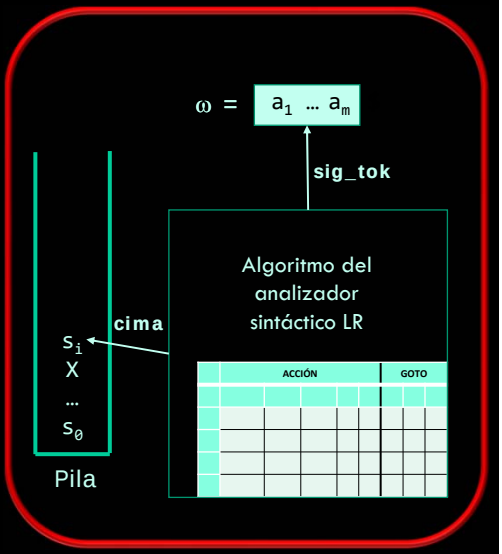
- Análizador Sintáctico Descendente Recursivo
  1. Diseñar en pseudo-código los procedimientos de cada símbolo No Terminal
  2. Implementar dichos procedimientos

```
...
7. F → ( E )
8. F → id
...
```

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


Implementación del Analizador Sintáctico

- Análizador 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



Implementación del Analizador Sintáctico

- Ficheros de salida
  - Formatos: https://dlsiis.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://dlsiis.fi.upm.es/procesadores/Herramientas.html
  - A partir de la gramática y del *parse* generado, construye el árbol correspondiente al programa fuente