Documento anónimo

2.pdf

examenes finales

3° Procesadores de Lenguajes

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

EXAMEN ANÁLISIS LÉXICO Y TABLA DE SÍMBOLOS, 10 ENERO 2018

Paso 1: Identificación de *tokens*

<ID, posTS>: representa a las variables; el atributo es la posición en la tabla de símbolos donde está la variable
<ENT, val>: representa a las constantes enteras, siendo el atributo el valor del número
<MAYOR, ->: representa al operador >
<2MAYOR, ->: representa al operador >>
<MENOR, ->: representa al operador <
<2MENOR, ->: representa al operador <<
<CADENA, cad>: representa a las cadenas, siendo el atributo la propia cadena

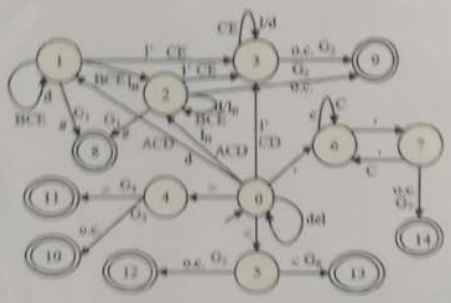
Paso 2: Gramática regular:

$A \rightarrow d\,B \mid l_H\,C \mid l'\,D \mid >E \mid <F \mid `G \mid del\,A$
$B \rightarrow d\,B \mid l_H\,C \mid l'\,D \mid \#$
$C \rightarrow d\,C \mid l_H\,C \mid l'\,D \mid \# \mid \lambda$
$D \rightarrow d\,D \mid l\,D \mid \lambda$

$E \rightarrow > \mid \lambda$
$F \rightarrow < \mid \lambda$
$G \rightarrow c\,G \mid `H$
$H \rightarrow `G \mid \lambda$

Donde: d::= dígitos l_H::= letras hexadecimales l::= letras l'::= l - l_H
del::=delimitador c::= cualquier carácter menos las comillas

Paso 3: construcción del AFD



Paso 4: Acciones Semánticas

Nombre	Descripción
A	núm:= valor(car_leído)
B	núm:= núm * 16 + valor(car_leído)
C	Concat (palabra)
D	cont:= 1
E	cont:= cont + 1
G_1	If (núm<2^{47}) Then GenToken (ENT, núm) Else Error (Número fuera de rango)

G_2	If (cont<81) Then
	p:= BuscaTS (palabra)
	If (p=null) Then p:= InsertaTS (palabra)
	GenToken (ID, p)
	Else Error (Variable con más de 80 caracteres)

G_3	GenToken (MAYOR, -)
G_4	GenToken (2MAYOR, -)
G_5	GenToken (MENOR, -)
G_6	GenToken (2MENOR, -)
G_7	GenToken (CADENA, palabra)
L	Se lee en todas las transiciones excepto en las de o.c.
Error	Todas las transiciones no indicadas producirían una acción de Error

else tipo_error

PROCESADORES DE LENGUAJES

SOLUCIÓN ANÁLISIS SINTÁCTICO

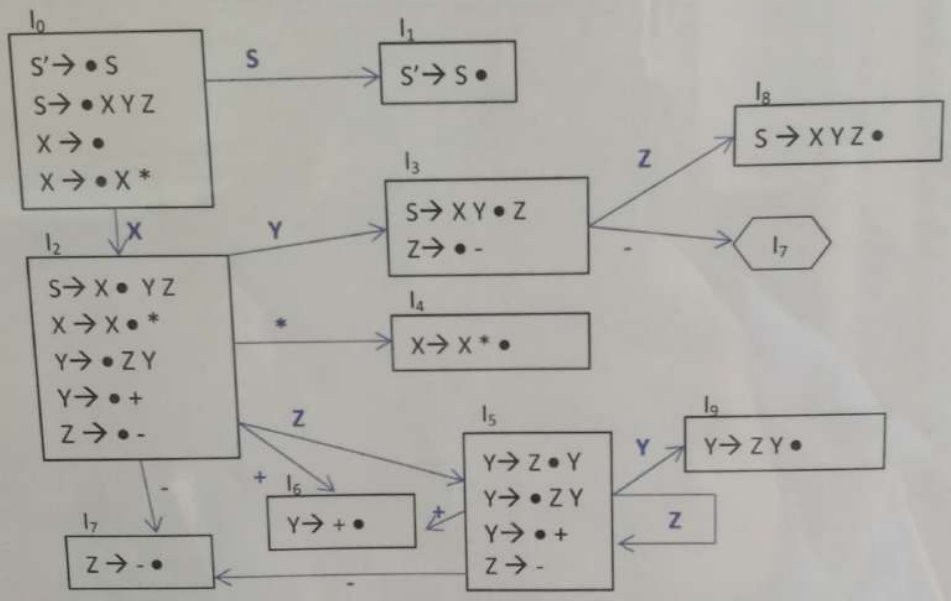
10 de enero de 2018

Sea la gramática G cuyo conjunto de reglas es:

P= { S→X Y Z
Y→Z Y |+
Z→-
X→ λ | X *

Se pide:

A1) Construye el Autómata Reconocedor de Prefijos Viables para esta gramática.



A2) Analiza los posibles conflictos a la hora de generar un Analizador LR para esta gramática.

En el autómata no aparece ningún estado con dos reducciones ni con reducción y desplazamiento, así que no hay posibilidad de que existan conflictos.

B1) Justifica si esta gramática es LL o no. En este último caso transfórmala para que lo sea (cambiando exclusivamente las reglas del No Terminal implicado)

X tiene una regla recursiva por la izquierda. Cambiamos las reglas de X para eliminar la recursividad: X→ λ | * X (reglas equivalentes a las anteriores)

Comprobamos la condición LL(1)

X→ λ | * X First (λ)={ λ} First (* X)={ *} Intersección vacía. Como aparece λ:
Follow (X)={ -, + } First (* X)={ *} Intersección vacía

Y→Z Y |+ First (Z Y)={ - } First (+)={ + } Intersección vacía

Por lo tanto la nueva gramática es LL

B2) Construye la Tabla de Decisión del Analizador Descendente no recursivo LL para la gramática del punto anterior (sigue el mismo orden de símbolos)

	*	+	-	$
S	S→X Y Z	S→X Y Z	S→X Y Z	
X	X→ * X	X→ λ	X→ λ	
Y		Y→ +	Y→Z Y	
Z			Z→ -	

PROCESADORES DE LENGUAJES

10 de enero de 2018

Observaciones: 1. Las calificaciones se publicarán hacia el 22 de enero.
2. La revisión será hacia el 24 de enero.
3. En la web se avisarán las fechas exactas.
4. La duración de este examen es de 40 minutos.

3. Sea un lenguaje con las siguientes características:

- Tiene los tipos int, long, byte, void, float, boolean y char. Esto quiere decir que existen reglas para los tipos (T → int | long | byte | void | float | boolean | char).
- Tiene expresiones (E → id | E op-ar E | E op-rel E | E op-lóg E | (E) | id [E] | cte_entera | cte_real | ...).
- Exige declaración previa de variables
- Las variables y los vectores pueden ser de cualquier tipo del lenguaje, salvo void.
- Los tipos int, long y byte son variantes del tipo "entero" y son compatibles entre sí. Para el resto de tipos, el lenguaje no tiene conversión automática entre ellos.
- Al asignar valores a un vector completo (id={L}), para cada elemento del vector tiene que haber uno y solo un valor.
- La variable índice (id) de la sentencia from se declara en la propia sentencia y ha de ser de alguno de los tres tipos "entero".
- La sentencia from funciona de la siguiente manera: se realiza la asignación id=E; si la expresión del when es cierta, se ejecuta el cuerpo del from y se vuelve al inicio; si es falsa, se sale del from.

Se pide diseñar el Analizador Semántico mediante un Esquema de Traducción únicamente para las siguientes reglas de la gramática (usando el espacio proporcionado para cada regla):

```
D → { zona_decl := true } T id                    /* declaración de una variable */
{ if T.tipo ≠ vacío
  then InsertarTipoTS(id.entrada, T.tipo)
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño
  else error()  /* Las variables no pueden ser de tipo void */
zona_decl := false }
```

```
D → { zona_decl := true }              /* declaración de un vector de "cte_entero" elementos */
    T id [ 1 .. cte_entera ]
{ if T.tipo ≠ vacío
  then InsertarTipoTS(id.entrada, array(1..cte_entera.val, T.tipo))
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño * cte_entera.val
  else error()  /* Los elementos de un vector no pueden ser de tipo void */
zona_decl := false }
```

```
S → id = E                                        /* asignación de un valor a una variable */
{ id.tipo:= BuscaTipoTS(id.entrada)
  S.tipo:= if (id.tipo = E.tipo ≠ tipo_error) OR
              (id.tipo ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
           then tipo_ok
           else tipo_error  /* tipos incompatibles en la asignación */ }
```

```
S → id [ cte_entera ] = E                  /* asignación de un valor a un elemento del vector */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array(1..n, t)
           then if E.tipo = t OR
                   (t ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
                then if 0 < cte_entera.val ≤ n
                     then tipo_ok
                     else tipo_error  /* el vector id tiene solo "n" elementos */
                else tipo_error /* la expresion no es del tipo de los elementos del vector */
           else tipo_error  /* la variable id no es un vector */}
```

```
S → id = { L }                             /* asignación de valores a un vector completo */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array (1..n, t)
           then if L.tipo = t OR
                   (t ∈ {int, long, byte} AND L.tipo ∈ {int, long, byte})
                then if L.núm_elem = n
                     then tipo_ok
                     else tipo_error
                          /* Tiene que haber uno y solo un valor para cada elemento del vector */
                else tipo_error
```

```
{ zona_decl := true } Tid
{ if T.tipo ≠ vacio
  then InsertarTipoTS(id.entrada, T.tipo)
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño
  else error()  /* las variables no pueden ser de tipo void */
zona_decl := false }
```

```
D → { zona_decl := true }                    /* declaración de un vector de "cte_entera" elementos */
    Tid [ 1 .. cte_entera ]
{ if T.tipo ≠ vacio
  then InsertarTipoTS(id.entrada, array(1..cte_entera.val, T.tipo))
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño * cte_entera.val
  else error()  /* los elementos de un vector no pueden ser de tipo void */
zona_decl := false }
```

```
S → id = E                                   /* asignación de un valor a una variable */
{ id.tipo:= BuscaTipoTS(id.entrada)
  S.tipo:= if (id.tipo = E.tipo ≠ tipo_error) OR
              (id.tipo ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
           then tipo_ok
           else tipo_error  /* tipos incompatibles en la asignación */ }
```

```
S → id [ cte_entera ] = E                    /* asignación de un valor a un elemento del vector */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array(1..n, t)
           then if E.tipo = t OR
                   (t ∈ {int, long, byte} AND E.tipo ∈ {int, long, byte})
                then if 0 < cte_entera.val ≤ n
                     then tipo_ok
                     else tipo_error  /* el vector id tiene solo "n" elementos */
                else tipo_error /* la expresión no es del tipo de los elementos del vector */
           else tipo_error  /* la variable id no es un vector */}
```

```
S → id = { L }                               /* asignación de valores a un vector completo */
{ S.tipo:= if BuscaTipoTS(id.entrada) = array (1..n, t)
           then if L.tipo = t OR
                   (t ∈ {int, long, byte} AND L.tipo ∈ {int, long, byte})
                then if L.núm_elem = n
                     then tipo_ok
                     else tipo_error
                          /* Tiene que haber uno y solo un valor para cada elemento del vector */
                else tipo_error
                          /* los valores asignados no son del tipo de los elementos del vector */
           else tipo_error  /* La variable id no es un vector */ }
```

```
L → E { L.tipo:= E.tipo                      /* una expresión */
      L.núm_elem:= 1 }
```

```
L → E , L1                                   /* una lista de expresiones */
{ L.tipo:= if E.tipo = L1.tipo OR
             (E.tipo ∈ {int, long, byte} AND L1.tipo ∈ {int, long, byte})
           then E.tipo
           else tipo_error  /* todos los elementos han de ser del mismo tipo (o "entero") */
  L.núm_elem:= 1 + L1.núm_elem }
```

```
S → from { zona_decl := true }  Tid          /* sentencia repetitiva */
{ if T.tipo ∈ {int, long, byte}
  then InsertarTipoTS(id.entrada, T.tipo)
       InsertarDesplTS(id.entrada, despl)
       despl:= despl + T.tamaño
  else error()  /* tipo erróneo en el índice del from. Debe ser de algún tipo "entero" */ }
zona_decl := false }
   = E1 when E2 { S1 }
{ S.tipo:= if T.tipo ∈ {int, long, byte}
           then if E1.tipo ∈ {int, long, byte}
                then if E2.tipo = lógico
                     then S1.tipo
                     else tipo_error  /* la expresión del when debe ser de tipo lógico */
                else tipo_error /* el valor asignado a id no es de uno de los tipos "entero" */
           else tipo_error  /* declaración errónea en la sentencia from */ }
```

PROCESADORES DE LENGUAJES

SOLUCIÓN ANÁLISIS SINTÁCTICO

10 de enero de 2018