E79 Procesadores de lenguaje

Examen de teoría (2 de junio de 2001 (solución))

Esta es una de las posibles soluciones al examen. Como puedes imaginar existen muchas otras igual de correctas, tómate esta como una orientación.

PREGUNTA 1 (6 PUNTOS)

Analizador léxico

Dado que las especificaciones de categorías léxicas y los segmentos de código Python pueden aparecer en cualquier parte, los trataremos como si fuesen comentarios. Las correspondientes acciones almacenarán la información en variables globales.

Las categorías serán:

Categoría	Expresión regular	Acciones
espacios	`[␣\t\n]+`	Omitir
comentarios	`/\*([^*]|\*[^/])*\*/`	Omitir
abre	`{`	Emitir.
cierra	`}`	Emitir.
dosp	`:`	Emitir.
barra	`|`	Emitir.
terminal	`[a-zA-Z]+`	Copiar lexema, emitir.
noterminal	`<[a-zA-Z]+>`	Copiar lexema sin < ni >, emitir.
acción	`@[^\n@]*@`	Copiar lexema sin `@`, emitir.
código	`\n%([^\n]|\n[^%])*\n%`	Añadir lexema sin `\n%` a la lista de código, omitir.
esléxica	`{[a-zA-Z]+|[a-zA-Z_][0-9a-zA-Z_]*|[^\n}]*}`	Separar las partes, comprobar la expresión, añadir a la lista de especificaciones omitir

Las únicas categorías que tienen atributos son **terminal**, **noterminal** y **acción**. Su atributo es el lexema, adecuadamente modificado. Al atributo de **noterminal** y **terminal** lo llamaremos *nombre* y al de **acción**, *código*. Dado que en el enunciado no se dice nada, se puede decidir si el lexema lo guardamos transformando todas las letras a minúsculas o mayúsculas o lo dejamos como está.

Gramática

Escribiremos una GPDR. Por la elección del analizador léxico, no es necesario preocuparse de los segmentos de código Python ni de las especificaciones de categorías léxicas.

$$
\begin{aligned}
\langle \text{Especificación} \rangle &\rightarrow \langle \text{Regla} \rangle (\langle \text{Regla} \rangle)^* \\
\langle \text{Regla} \rangle &\rightarrow \textbf{abre noterminal dosp } \langle \text{ParteDerecha} \rangle (\textbf{barra } \langle \text{ParteDerecha} \rangle)^* \textbf{ cierra} \\
\langle \text{ParteDerecha} \rangle &\rightarrow (\textbf{terminal}|\textbf{noterminal}|\textbf{acción}|\langle \text{Regla} \rangle)^*
\end{aligned}
$$

Para comprobar que es RLL(1), comenzamos por calcular los primeros:

Elemento	Primeros
⟨Especificación⟩	**abre**
⟨Regla⟩	**abre**
(⟨Regla⟩)*	**abre**, λ
⟨ParteDerecha⟩	**abre**,**terminal**, **noterminal**, **acción**, λ
(**barra** ⟨ParteDerecha⟩)*	**barra**, λ
(**terminal**|**noterminal**|**acción**|⟨Regla⟩)*	**abre**, **terminal**, **noterminal**, **acción**, λ
terminal|**noterminal**|**acción**|⟨Regla⟩	**abre**, **terminal**, **noterminal**, **acción**

Calculamos los siguientes de los elementos que pueden anularse:

Elemento	Siguientes
(⟨Regla⟩)*	**$**
⟨ParteDerecha⟩	**barra**, **cierra**
(**barra** ⟨ParteDerecha⟩)*	**cierra**
(**terminal** |**noterminal** |**acción** |⟨Regla⟩)*	**cierra**, **barra**

Con esto llegamos a la tabla:

	abre	**cierra**	**dosp**	**barra**	**terminal**	**noterminal**	**acción**	**$**
⟨Especificación⟩	⟨Regla⟩(⟨Regla⟩)*							
⟨Regla⟩	(1)							
(⟨Regla⟩)*	⟨Regla⟩(⟨Regla⟩)*							λ
⟨ParteDerecha⟩	(2)	(2)		(2)	(2)	(2)	(2)	
(**barra** ⟨ParteDerecha⟩)*		λ		(3)				
(2)	(4)	λ		λ	(4)	(4)	(4)	
(5)	⟨Regla⟩				**terminal**	**noterminal**	**acción**	

Hemos utilizado las siguientes abreviaturas:

- (1): **abre noterminal dosp** ⟨ParteDerecha⟩(**barra** ⟨ParteDerecha⟩)* **cierra**
- (2): (**terminal**|**noterminal**|**acción**|⟨Regla⟩)*.
- (3): (**barra**⟨ParteDerecha⟩)(**barra**⟨ParteDerecha⟩)*.
- (4): (**terminal**|**noterminal**|**acción** |⟨Regla⟩)(**terminal**|**noterminal**|**acción**|⟨Regla⟩)*.
- (5): (**terminal**|**noterminal**|**acción**|⟨Regla⟩)

Así pues, no hay conflictos y la gramática es RLL(1).
Las comprobaciones semánticas hay que realizar sobre la entrada son:

- Que todo terminal utilizado en las reglas haya aparecido en una especificación de categoría léxica.
- Que todo no terminal aparezca en, al menos, una parte izquierda.

Dependiendo de las asunciones que se hagan, se puede además comprobar para dar un *warning*:

- Que todo terminal especificado aparezca en alguna regla.
- Que todo no terminal distinto del inicial aparezca en una parte derecha.

Sin embargo, no hace falta comprobar el código Python ni que la gramática es LR(1) (se supone que lo hace la biblioteca proporcionada).

Como representación intermedia, vamos a mantener tres listas:

- Una lista con el código Python obtenido de los segmentos.
- Una lista con las categorías léxicas que se han ido especificando.

- Una lista con las reglas de la gramática. Estas reglas serán pares formados por un no terminal y una lista representando la parte izquierda.

Las dos primeras listas las mantiene el analizador léxico. La gramática atribuida para obtener la tercera es:

⟨Especificación⟩ → {lreglas:=∅} ⟨Regla⟩$_1$ {lreglas:= lreglas+⟨Regla⟩$_1$.lreglas}
(⟨Regla⟩$_2$ {lreglas:= lreglas+⟨Regla⟩$_2$.lreglas})*

⟨Regla⟩ → {⟨Regla⟩.lreglas:=∅} **abre noterminal dosp**
{⟨Regla⟩.izda:= `mcNTerm`(**noterminal**.nombre); ⟨ParteDerecha⟩$_1$.izda:= ⟨Regla⟩.izda}
⟨ParteDerecha⟩$_1$ {⟨Regla⟩.lreglas:=⟨Regla⟩.lreglas+⟨ParteDerecha⟩$_1$.lreglas}
(**barra** {⟨ParteDerecha⟩$_2$.izda:= ⟨Regla⟩.izda}
⟨ParteDerecha⟩$_2$ {⟨Regla⟩.lreglas:=⟨Regla⟩.lreglas+⟨ParteDerecha⟩$_2$.lreglas})* **cierra**

⟨ParteDerecha⟩ → {⟨ParteDerecha⟩.lreglas:=∅; l:=∅}
(**terminal** {l:= l+`mcTerm`(**terminal**.nombre)}|
noterminal {l:= l+`mcNTerm`(**noterminal**.nombre)}|
acción {l:= l+`mcAccion`(**acción**.código)}|
⟨Regla⟩ {l:= l+⟨Regla⟩.izda; ⟨ParteDerecha⟩.lreglas:=⟨ParteDerecha⟩.lreglas+⟨Regla⟩.lreglas})*
{⟨ParteDerecha⟩.lreglas:= ⟨ParteDerecha⟩.lreglas+(⟨ParteDerecha⟩.izda, l)}

En lreglas tenemos la lista que vamos construyendo. El atributo lreglas de ⟨Regla⟩ guardará las listas que se encuentren en su expansión. Además, para facilitar el tratamiento de las partes derechas, tendrá un atributo que contenga su parte izquierda. La construcción de las reglas se realiza en ⟨ParteDerecha⟩. Su atributo lreglas contendrá las reglas anidadas que encuentre y la que construirá. Para construirla tiene el atributo izda que le dice qué no terminal tiene en la parte izquierda.

Para generar código, podemos utilizar el siguiente algoritmo:

Algoritmo generación de código
  para $a \in$ lista de código Python **hacer**
    emitir(a);
  fin para
  emitir(`'from metacompilador import mcAnalizadorLexico, mcAnalizadorSintactico,`
    `mcTerm, mcNTerm, mcAccion\n'`)
  emitir(`'alex= mcAnalizadorLexico()\n'`)
  para (nombre, acción, expresión)$\in$ lista de especificaciones léxicas **hacer**
    emitir(`'alex.especifica(%s,%s,%s)\n'`,nombre, acción, expresión)
  fin para
  emitir(`Analizador= mcAnalizadorSintactico(alex)`)
  para (izda,dcha)$\in$ lreglas **hacer**
    emitir(`'Analizador.regla(%s,['`, izda);
    para $i \in$ dcha **hacer**
      emitir(i,`','`);
    fin para
    borrar(`','`);
    emitir(`'])\n'`);
  fin para
  emitir(`'Analizador.compila()\n'`)
  emitir(`'main()\n'`)
Fin

Pregunta 2 (2 puntos)

Podemos seguir una estrategia similar a la presentada en los apuntes para calcular los primeros:

Algoritmo segundos(α)
  $C = \emptyset$;
  si $\alpha = \mathtt{a}\beta$ **entonces**
    $C :=$ primeros(β) $- \{\lambda\}$;
  si no si $\alpha = \langle \mathsf{A} \rangle \beta$ **entonces**
    si $\lambda \in$ primeros($\langle \mathsf{A} \rangle$) **entonces**
      $C :=$ segundos(β);
    fin si
    si unitario($\langle \mathsf{A} \rangle$) **entonces**
      $C := C \cup$ primeros(β) $- \{\lambda\}$;
    fin si
    para todo $\langle \mathsf{A} \rangle \to \gamma \in P$ **hacer**
      $C := C \cup$ segundos(γ);
    fin para
  fin si
  devolver C;
fin primeros

Pregunta 3 (2 puntos)

1) Hay un conflicto desplazamiento/reducción en el primer estado. Por la regla $\langle \mathsf{A} \rangle \to \langle \mathsf{B} \rangle \mathtt{c} \langle \mathsf{D} \rangle$ vemos que c es un siguiente de $\langle \mathsf{B} \rangle$, así que la gramática no es SLR.
2) El segundo estado está mal construido; faltan los ítems correspondientes a las reglas de $\langle \mathsf{D} \rangle$.
3) Para saber si hay conflicto, tenemos que averiguar si c es o no un siguiente de $\langle \mathsf{A} \rangle$, pero no tenemos información suficiente con los estados que se ven.
4) A partir del segundo estado, vemos que $\langle \mathsf{B} \rangle \Rightarrow \langle \mathsf{C} \rangle \Rightarrow \lambda$, lo que unido a la regla $\langle \mathsf{B} \rangle \to \langle \mathsf{A} \rangle \langle \mathsf{B} \rangle \mathtt{c}$, nos dice que c es un siguiente de $\langle \mathsf{A} \rangle$. Así pues, tenemos un conflicto en el primer estado.