E79 Procesadores de lenguaje

Examen de teoría (30 de junio de 2003 (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)

Norma de los vectores

Idea general

Introduciremos el operador norma en el nivel más prioritario de la gramática. El código que se genere dejará el resultado de la norma en el tope de la pila. Para hacer el cálculo, utilizaremos un bucle en tiempo de ejecución.

Nivel léxico

Nos basta con introducir la categoría léxica **db** (doble barra). Su expresión regular es \|\| y sus componentes no tienen ningún atributo. La acción asociada a ella es emitir.

Nivel sintáctico

Como hemos comentado arriba, basta con añadirla en el nivel más prioritario. Bastaría una regla como:

⟨Base⟩ → **db id db**

La presencia de **db** al principio de la regla hace que no haya conflictos.

Nivel semántico

Representaremos la norma mediante el nodo NodoNorma. El esquema de traducción se modifica fácilmente:

⟨Base⟩ → **db id db** {⟨Base⟩.árbol:= NodoNorma(**id**.lexema)}

El método que realiza las comprobaciones semánticas de `nodoNorma` comprueba que el identificador está en la tabla de símbolos y es un vector. Como tipo propio devuelve real.

Generación de código

Tendremos que crear un bucle que calcule la norma. Podemos aprovechar la pila de flotantes para los resultados intermedios y utilizar una dirección fija (en nuestro caso 2) para el contador que recorrerá el vector. La generación de código quedaría algo así como se ve en la figura 1.

Hemos supuesto que v es el atributo que contiene el identificador del vector, que base es la dirección de su primer elemento y que talla el número de elementos. No hace falta restar uno cuando se suman base y talla porque el contador se incrementa antes del test.

Información sobre vectores

Idea general

Dado que la información que vamos a utilizar es conocida en tiempo de compilación, bastará con sustituir en el código las consultas por los números (enteros) correspondientes.

Nivel léxico

Aparece un componente nuevo, el punto, y tres nuevas palabras reservadas. La categoría **punto** tiene como expresión regular \., ningún atributo asociado y su acción es emitir. En cuanto a las nuevas palabras

BORRADOR

```
Objeto NodoNorma:
    ...
    Método genera_código()
        et:=nueva_etiqueta();
        emite (&2=, tsímbolos[v].base);
        emite (FPUSH 0.0);
        si tsímbolos[v].tipo_base= entero entonces
            emite (et: PUSH 0[&2]);
            emite (TOFLOAT);
        si no
            emite (et: FPUSH 0[&2]);
        fsi
        emite (FDUP);
        emite (FMUL);
        emite (FADD);
        emite (INC &2);
        emite (PUSH tsímbolos[v].base+tsímbolos[v].talla);
        emite (!=);
        emite (BRD et);
        emite (FSQRT);
    fin genera_código
    ...
fin NodoNorma
```

Figura 1: Algoritmo de generación de código para la norma de los vectores.

reservadas, siguen las normas de MM3, así que sus expresiones regulares son `talla|TALLA`, `inf|INF`, `sup|SUP`. Tampoco tienen atributos asociados y se emiten.

Nivel sintáctico

Tendremos que introducir los accesos como "decoraciones" de los identificadores que aparecen en las expresiones, junto a los paréntesis para llamadas a funciones o los corchetes de los vectores. Esta parte de la gramática queda parecida a:

$$\begin{aligned} \langle \text{Base} \rangle &\rightarrow \textbf{id}\ \langle \text{Decoraciones} \rangle \\ \langle \text{Decoraciones} \rangle &\rightarrow \lambda \mid\ [\langle \text{Expresión} \rangle] \mid\ (\langle \text{Parámetros} \rangle) \mid \textbf{punto}\ (\textbf{talla}|\textbf{inf}|\textbf{sup}) \end{aligned}$$

Nuevamente, es fácil ver que no se introducen conflictos.

Nivel semántico

Para ahorrarnos complicaciones, en este nivel "haremos trampa" y sustituiremos los accesos por constantes enteras. En el esquema de traducción:

$$\begin{aligned} \langle \text{Base} \rangle &\rightarrow \textbf{id}\ \{\langle \text{Decoraciones} \rangle.\text{id} := \text{id.lexema}\}\langle \text{Decoraciones} \rangle\ \{\langle \text{Base} \rangle.\text{árbol} := \langle \text{Decoraciones} \rangle.\text{árbol}\} \\ \langle \text{Decoraciones} \rangle &\rightarrow \textit{punto}\ \{\textbf{si}\ \text{tsímbolos}[\langle \text{Decoraciones} \rangle.\text{id}].\text{tipo} != \text{vector}\ \textbf{entonces}\ \textit{error};\ \} \\ &\quad (\ \textbf{talla}\ \{\langle \text{Decoraciones} \rangle.\text{árbol} := \text{NodoEntero}(\text{tsímbolos}[\langle \text{Decoraciones} \rangle.\text{id}].\text{talla})\} \\ &\quad \mid \textbf{inf}\ \{\langle \text{Decoraciones} \rangle.\text{árbol} := \text{NodoEntero}(\text{tsímbolos}[\langle \text{Decoraciones} \rangle.\text{id}].\text{sup})\} \\ &\quad \mid \textbf{sup}\ \{\langle \text{Decoraciones} \rangle.\text{árbol} := \text{NodoEntero}(\text{tsímbolos}[\langle \text{Decoraciones} \rangle.\text{id}].\text{inf})\} \\ &\quad) \end{aligned}$$

Generación de código

Con la estrategia utilizada, no es necesario cambiar la generación de código.

BORRADOR

Interrupción de bucles

Idea general

La idea general de esta modificación consiste en tener por cada bucle dos etiquetas, una que indica dónde saltar ante un `termina` y otra dónde saltar ante un `reinicia`. El código de las sentencias será simplemente un salto a la correspondiente etiqueta.

Nivel léxico

Introducimos las nuevas palabras reservadas. Sus expresiones regulares son `termina`|`TERMINA` y `reinicia`|`REINICIA`. No tienen atributos y la acción asociada es emitir.

Nivel sintáctico

Son nuevas sentencias, así que las asociamos al no terminal ⟨Sentencia⟩. Las reglas correspondientes son triviales:

⟨Sentencia⟩ → **reinicia** ;
⟨Sentencia⟩ → **termina** ;

La comprobación de que están dentro de un bucle se deja al nivel semántico.

Nivel semántico

Para las comprobaciones semánticas, utilizaremos una variable global que nos indica si estamos o no en un bucle. Es fácil de mantener[1]. Una opción es que la variable sea entera y que al comenzar a analizar un bucle se incremente su valor. Cuando se sale del bucle se decrementa. Alternativamente, la variable puede ser booleana y al entrar en el bucle se guarda el valor y se pone a cierto. Al salir se restaura el valor que tenía. En cualquier caso, el nivel semántico se limita a comprobar con ayuda de esta variable si las sentencias que nos ocupan están en un bucle y emite los correspondientes nodos (`NodoTermina` y `NodoReinicia`).

Generación de código

Aquí utilizaremos las variables globales que mencionábamos al principio. En el caso del bucle `mientras` hay que generar una etiqueta que apunte a la siguiente instrucción y utilizarla como etiqueta para `termina`. Es posible que esta etiqueta ya se generara de todas formas. En el bucle `repite`, dependerá de si se utiliza o no la pila para almacenar el límite del bucle, un contador o alguna otra cosa. Si no se emplea, se genera una etiqueta directamente al final y ya está. En caso contrario, hay que generar una etiqueta a un segmento de código donde se hagan los `POP` necesarios.

Como el párrafo anterior queda lioso por ser demasiado general, pondremos dos ejemplos concretos:

- Para el bucle `repite` se utilizan variables creadas al efecto: la etiqueta de salida apunta directamente a la siguiente instrucción.
- Se utiliza la pila para guardar el número de iteraciones: la etiqueta de salida apunta al sitio donde se hace el `POP` que elimina ese contador.

En cuanto a `reinicia`, habrá que generar una etiqueta justo delante de la comprobación de la condición del `mientras` (que es probable que ya se haga) y otra delante del incremento de la variable en el caso del `repite`.

Con esto, la generación de código es trivial:

- Para `termina` se genera la instrucción `JMP` a la etiqueta de terminación.
- Para `reinicia` se genera la instrucción `JMP` a la etiqueta de reinicio.

[1] De hecho, es lo mismo que hay que hacer para saber si `contador` aparece o no en un `repite`.

BORRADOR

Pregunta 2 (2 puntos)

- Cadenas de ceros y unos que representan caracteres ASCII imprimibles (7 bits con valor entre 32 y 127).

  La expresión regular es fácil:

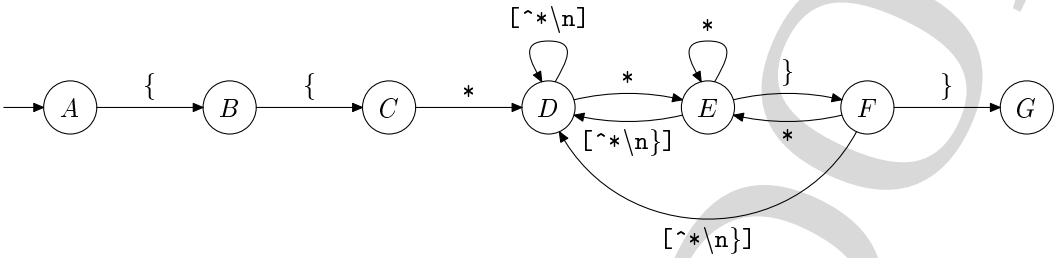
  `01(0|1)(0|1)(0|1)(0|1)(0|1)|1(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)`

  También se admitiría que no fuesen exactamente siete bits:

  `0?1(0|1)(0|1)(0|1)(0|1)(0|1)|1(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)`

- Comentarios de un lenguaje de programación que empiezan por la secuencia `{{*`, terminan con `*}}` y no contienen ni saltos de línea, ni ninguna secuencia `*}}`.

  Esta es más fácil con un autómata:



  La expresión regular viene a ser:

  `{{\*([^*\n]|\*(}?\*)*(}?[^*}\n]))*\*(}?\*)*}}`

- Cadenas formadas únicamente por los tokens **id** (identificador), **entrada** (palabra clave `entrada`), **corcheteAb** (corchete abierto), **corcheteCer** (corchete cerrado), **flecha** (la secuencia `->`), **pyc** (punto y coma) y que son sentencias válidas en MM3.

  Esto es imposible de expresar con una expresión regular porque los corchetes tienen que estar equilibrados.

- Literales de cadena de MM3 con, al menos, una secuencia de escape.

  Podemos usar una expresión regular como:

  `"([^"$\n]|$[^\n])*$(n|t|"|$)([^"$\n]|$[^\n])*"`

  Nota: siendo estrictos, la expresión regular debería limitar la talla de la cadena (no del literal) a 80 caracteres. Para eso podría usarse algo así como:

  `"$(n|t|"|$)([^"$\n]|$[^\n])`^{79}`"| "([^"$\n]|$[^\n])$(n|t|"|$)([^"$\n]|$[^\n])`^{78}`"|`
  `"([^"$\n]|$[^\n])`^{2}`$(n|t|"|$)([^"$\n]|$[^\n])`^{77}`"| ... "([^"$\n]|$[^\n])`^{79}`$(n|t|"|$)"`

Pregunta 3 (2 puntos)

Como comenté durante el examen, debe entenderse que m es la talla de la parte derecha más larga. Supongamos ahora que la gramática es LR(0). Vamos a fijarnos en dos cosas:

- Dado que en cada regla el punto aparece delante de cada uno de los símbolos de la parte derecha, tiene que haber al menos m estados en el autómata. Esto es, si la regla con la parte derecha más larga es $\langle A \rangle \to X_1 \dots X_m$ tendremos un estado por cada uno de los ítems $\langle A \rangle \to \cdot X_1 X_2 \dots X_m$, $\langle A \rangle \to X_1 \cdot X_2 \dots X_m, \dots$, $\langle A \rangle \to X_1 X_2 \dots \cdot X_m$.
- Por otro lado, tiene que haber un estado donde se reduzca cada uno de los no terminales, lo que son n estados más. Aquí es donde interviene que la gramática sea LR(0); no puede haber dos ítems que pidan reducción en el mismo estado.

Los dos conjuntos de estados son disjuntos (en el primero no hemos incluido el ítem con el punto al final que es que pide la reducción), así que tenemos al menos $m + n$ estados.

¿Qué pasa si la gramática es SLR? El primero punto de antes sigue sirviendo, pero el segundo no, ya que sí que puede haber estados con distintas reducciones. Pero podemos darnos cuenta de que todo no terminal de la gramática original aparece a la izquierda del punto en algún estado (o, dicho de otra forma, aparece como entrada en alguna transición). Esto son $n - 1$ estados. Además tenemos el estado inicial, con lo que son n. Lo que sucede es que, a diferencia de antes, los dos grupos no son necesariamente disjuntos, así que sólo podemos afirmar que hay máx(m, n) estados.

BORRADOR