E79 Procesadores de lenguaje

Examen de teoría (1 de junio de 2002)

PREGUNTA 1 (6 PUNTOS)

A continuación, se presentan cuatro posibles extensiones del lenguaje 2KS. Elige tres de ellas y explica claramente qué modificaciones se tendrían que hacer en un compilador de 2KS para que las aceptara.

En tu descripción de las modificaciones, procura ser claro, escueto y preciso. En particular, no es necesario que describas partes del compilador que no estén afectadas por las modificaciones. Puedes optar por descripciones algorítmicas o en lenguaje natural para lograr una mayor sencillez en la explicación. También puede facilitarte la exposición, una estructura que siga las distintas etapas del compilador.

Explicita cualquier asunción que hagas acerca del compilador o del enunciado propuesto.

Definición de funciones con tipo vacío

Con esta extensión se permite la declaración de funciones que no devuelvan resultado alguno. Para ello, se introduce la nueva palabra reservada `BUIT`. Esta palabra podrá aparecer en la declaración de la función en el lugar donde se escribe el tipo del resultado de la función. Además, en el cuerpo de la función, podrá aparecer en la parte derecha de la asignación a `RESULTAT` (de hecho, es la única parte derecha posible en una función de este tipo). El resultado de una asignación de ese estilo es el retorno de la función, sin devolver ningún valor. Tampoco se devolverá valor alguno si la ejecución de la función no termina mediante una sentencia de retorno. Lógicamente, las llamadas a una función de tipo `BUIT` no pueden aparecer más que en sentencias expresión en las que la expresión consista únicamente en la llamada.

Bucles sobre vectores

Con esta extensión se introduce un nuevo tipo de bucle. La forma de este bucle es

`BUCLE` *i*`<- VALORS(`*v*`):` *sentencia*

donde v es un vector y i es una variable con un tipo igual o más general que el tipo base de v. La ejecución de este bucle supone la ejecución de la sentencia tantas veces como elementos tenga el vector, teniendo i en cada ocasión uno de los valores almacenados en v. Así, las líneas

```
t<- 0;
BUCLE i<- VALORS(v): t<- t+i*i;
```

dejarían en `t` la suma de los cuadrados de los valores almacenados en `v`.

Nota: el código generado no puede crecer exponencialmente con la talla del programa fuente. Es decir, dado un vector de diez componentes no se permite generar el código equivalente al que se generaría con diez asignaciones a la variable de control y diez copias de la sentencia.

Operatorios sobre vectores

Esta extensión introduce siete nuevos operadores: `[+]`, `[-]`, `[*]`, `[/]`, `[%]`, `[&]`, `[|]`. Estos operadores son similares a los utilizados en los "operatorios" estándar de 2KS, pero los valores que toma la variable muda son los almacenados en un vector. Así, la expresión `[+](i,a,i*i)` devuelve el resultado de calcular la suma de los cuadrados de los valores almacenados en el vector `a`. Como puedes ver en el ejemplo, la sintaxis es similar a la de los "operatorios" convencionales, pero en lugar del rango, se escribe un identificador de vector. Como en los "operatorios" convencionales, la variable muda del "operatorio" no puede aparecer como variable muda en otro "operatorio" anidado. Además, el tipo base del vector debe ser igual o menos general que el de la variable muda.

Es aconsejable que antes de plantearte cómo generarías código para estos operatorios, intentes generarlo a mano para un ejemplo concreto.

Nota: el código generado no puede crecer exponencialmente con la talla del programa fuente. Es decir, dado un vector de diez componentes no se permite generar el código equivalente al que se generaría con una expresión de diez términos.

Definición de parámetros con valores por defecto

En esta extensión se cambian las cabeceras de las funciones de modo que permitan la especificación de valores por defecto para los parámetros. Los parámetros en la declaración de una función pasan a tener una parte opcional tras el tipo y el nombre que consta de un símbolo `<-`, un signo opcional y una constante compatible con el tipo del parámetro. Cuando en una llamada a la función no hay ninguna expresión en el lugar correspondiente a un parámetro con valor por defecto, el parámetro toma como valor el expresado en la declaración. Así, con la cabecera

```
SUBRUTINA f(ENTER a, ENTER b<-2, REAL r<-1.0) DE TIPUS ENTER:
```

Las llamadas `f(1)` y `f(1,2)` son equivalentes a `f(1,2,1.0)`. En caso de que en la definición haya un parámetro formal con valor por defecto, todos los siguientes deben tener a su vez valor por defecto.

Pregunta 2 (2 puntos)

Dada la siguiente gramática:

$$\begin{array}{lcl} \langle E \rangle & \rightarrow & \langle E \rangle + \langle T \rangle \mid \langle T \rangle \\ \langle T \rangle & \rightarrow & \langle T \rangle * \langle F \rangle \mid \langle F \rangle \\ \langle F \rangle & \rightarrow & \mathbf{id} \mid \mathbf{cte} \mid (\langle E \rangle) \mid (\langle E \rangle \end{array}$$

escribe las tablas de acciones y sucesores correspondientes a un analizador SLR. Resuelve los conflictos que se plantean de modo que una expresión con paréntesis sin cerrar se interprete como si esos paréntesis estuvieran cerrados al final. Es decir, de modo que **id*(cte+id*cte** se interprete como **id*(cte+id*cte)**.

Pregunta 3 (2 puntos)

Di si las siguientes afirmaciones son o no ciertas. Justifica tus respuestas.

- Si una gramática RLL(1) contiene, entre otras, las reglas $\langle A \rangle \rightarrow \mathbf{a}(\langle B \rangle)^* \mathbf{c}$ y $\langle B \rangle \rightarrow \mathbf{b}$, entonces $\langle B \rangle$ no es anulable.
- Si una gramática SLR contiene la regla $\langle A \rangle \rightarrow \langle A \rangle \langle A \rangle \mathbf{a}$, entonces `a` no es primero de $\langle A \rangle$.
- Si una gramática contiene las reglas $\langle A \rangle \rightarrow \mathtt{ab}$ y $\langle B \rangle \rightarrow \mathtt{ab}$, no puede ser LL(1).
- Si un estado del autómata de prefijos viables de una gramática contiene los ítems $\langle A \rangle \rightarrow \langle B \rangle \cdot \langle C \rangle$ y $\langle C \rangle \rightarrow \cdot \langle A \rangle \langle D \rangle$ y la gramática tiene las reglas $\langle A \rangle \rightarrow \mathtt{a}$ y $\langle C \rangle \rightarrow \mathtt{a}$, entonces la gramática no es SLR.