E79 Procesadores de lenguaje

Examen de teoría (3 de julio 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. Las modificaciones son independientes entre sí; no hace falta que consideres sus posibles interacciones.

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.

Operador coma

Esta extensión permite utilizar la coma como operador de manera similar al C. El operador coma introduce un nuevo nivel de prioridad por debajo del de la suma, resta y disyunción. La semántica del nuevo operador es sencilla: se evalúa la parte izquierda, se descarta el resultado (pero no los posibles efectos secundarios) y después se evalúa la parte derecha, que constituye el resultado final. El tipo del operador es el tipo de la expresión que tiene como parte derecha. La asociatividad es a izquierdas.

Este operador plantea un problema con la sintaxis de las llamadas a función. Para resolverlo, en el contexto de una llamada, la coma separa parámetros y si se quiere utilizar el operador hay que rodearlo de paréntesis. Así, `f(1,3)` es una llamada a `f` con dos parámetros y `f((1,3))` es una llamada con un parámetro que utiliza el operador coma (y que acaba siendo equivalente a `f(3)`).

Sobrecarga de funciones

Esta modificación permite que existan varias funciones con el mismo nombre en un programa siempre y cuando difieran en el número o tipo de sus parámetros (pero sin tener en cuenta el tipo del resultado). El método que se sigue para decidir a qué función se llama en un momento dado es el siguiente:

- En primer lugar se comprueba si hay alguna función tal que los tipos de los parámetros en la definición coincidan con los tipos de los argumentos en la llamada.
- Si no se encuentra ninguna función mediante en el paso anterior, se busca entre todas las funciones, por orden de defínción, la primera con tipos compatibles con los de la llamada.
- Si tampoco se encuentra ninguna función con el criterio anterior, se anuncia error.

Por ejemplo, supongamos que tenemos las siguientes cabeceras de función:

```
subrutina f(real i,real j) amb resultat enter;
subrutina f(enter i,enter j) amb resultat real;
subrutina f(enter i,real j) amb resultat enter;
```

y las variables `e1` y `e2` de tipo entero y `r1` y `r2` de tipo real. Entonces:

- `r1<- f(r1,e2);` es una llamada a la primera función.
- `r1<- f(e1,e2);` es una llamada a la segunda función.
- `e1<- f(e1,r2);` es una llamada a la tercera función.
- `e1<- f(e1,e2);` tiene un error de tipos (asignación de un real a un entero).
- Si además intentamos declarar

  ```
  subrutina f (enter i,real j) amb resultat real;
  ```

  tendremos un error porque los tipos de los parámetros coinciden con los de una definición anterior.

Operadores sobre vectores

Esta extensión introduce siete nuevos operadores: `[+]`, `[-]`, `[*]`, `[/]`, `[%]`, `[&]`, `[|]`. Estos operadores reciben como parámetros una lista de dos o más vectores, todos del mismo tamaño, separados por comas. Los operadores no producen ningún resultado; su utilidad reside en sus efectos secundarios: el primer vector almacena el resultado de sumar, restar, multiplicar, etc. los restantes vectores componente a componente. Así, la línea:

```
[+](a,b,c,d);
```

hace que la primera componente de `a` sea igual a la suma de las primeras componentes de `b`, `c` y `d`. La segunda componente será igual a la suma de las segundas componentes y así sucesivamente. Las restricciones que deben seguirse para utilizar estos operadores son:

- Los vectores deben tener el mismo tamaño, aunque no necesariamente los mismos límites inferior y superior.
- Los tipos base de los vectores deben seguir las normas habituales de 2KS. En particular, el tipo base del primer vector debe ser igual o más general que los tipos base de los siguientes vectores y, en el caso de los operadores `[%]`, `[&]` y `[|]`, los vectores deben ser de enteros.

Nota: el código generado no puede crecer exponencialmente con la talla del programa fuente. Es decir, si los vectores tienen diez componentes no se permite generar el código equivalente al que se generaría con diez asignaciones.

Bucle con contador automático

Esta extensión dota al lenguaje de una nueva estructura de control, un bucle con la sintaxis

`BUCLE` *expresión* `VOLTES:` *sentencia*

donde *expresión* es de tipo entero e indica el número de veces que ha de ejecutarse la sentencia que constituye el cuerpo del bucle. La expresión se ha de evaluar una sola vez, antes de la primera ejecución de sentencia (si es que ésta procede), e indica el número de iteraciones del bucle, salvo si el valor de la expresión es negativo, en cuyo caso se actuará como si el resultado de la evaluación hubiera sido cero.

Dentro del cuerpo de uno de estos bucles puede utilizarse una expresión simple de tipo entero, denotada mediante la palabra reservada `VOLTA`, que se evaluará al número de iteración que se esté ejecutando (a la primera iteración le corresponde el valor 1; a la segunda, 2; etcétera). En caso de bucles `VOLTES` anidados, la expresión `VOLTA` siempre hace referencia al número de iteración del más interior de éstos.

Por ejemplo, para escribir la lista de todas las posibles variaciones sin repetición de dos elementos del conjunto $\{1, 2, 3, 4\}$ se podría utilizar el siguiente programa:

```
enter i;
bucle 4 voltes:
  << i <- volta;
     bucle 4 voltes:
       si volta!=i: eixida <- i," ",volta,"@n";
  >>
```

Pregunta 2 (4 puntos)

Queremos enriquecer las GPDR con un nuevo operador: el operador lista (^). Este operador se define de modo que $(a\hat{}b)$ es igual a $a(ba)^*$. Así, la regla para describir una lista de identificadores separados por comas se escribe como:

$$\langle \mathsf{ID} \rangle \quad \rightarrow \quad \mathbf{id} \; \hat{} \; ,$$

Reescribe los algoritmos del tema 3 para el cálculo de primeros, anulables y siguientes de modo que tengan en cuenta este operador. Utilízalos para comprobar si la siguiente gramática es RLL(1):

$$\begin{array}{lcl} \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{T} \rangle \hat{} (+|-) \\ \langle \mathsf{T} \rangle & \rightarrow & \langle \mathsf{F} \rangle \hat{} (*|/) \\ \langle \mathsf{F} \rangle & \rightarrow & \mathbf{id}|\mathbf{cte}|(\langle \mathsf{E} \rangle) \end{array}$$