Pruebas puntuables del tema 5: generación de código

Procesadores de lenguaje - II26 - TE2

20 de mayo de 2010

Normativa En los siguientes ejercicios se plantean dos posibles extensiones del compilador Minicomp, partiendo de la versión inicial del curso 2009/2010. La duración total es de 60 minutos, a repartir entre ambas pruebas como creas conveniente. Puedes consultar libros y apuntes. No puedes utilizar dispositivos electrónicos para resolverlas.

Prueba 1 (0.50 puntos) Supongamos que queremos modificar Minicomp para que el lenguaje de entrada acepte una nueva operación que permita contar la cantidad de veces que aparece un valor entre los resultados de una serie de expresiones, empleando la sintaxis

`#(` *expresión*$_1$ `,` *expresión*$_2$ `,` ... `,` *expresión*$_N$ `)`

donde los símbolos `#(` de la apertura van unidos sin espacio entre ellos, las expresiones son de tipo entero, y $N \geq 1$. El resultado de la nueva operación es la cantidad de veces que el resultado de evaluar la primera expresión coincide con el resultado de evaluar cada una de las restantes (si no hay más expresiones, es cero). Se garantiza que las expresiones se evalúan en el orden en que aparecen de izquierda a derecha y que los efectos secundarios de evaluar cada una de las N expresiones se producen una sola vez. La nueva construcción es válida en cualquier contexto en que sea válida una expresión de tipo entero. Por ejemplo, la siguiente sentencia escribe 20:

```
escribe #(2 + 3, 4 + 1, 5 + #(6), 7, 8 * 9) * 10;
```

Imagina que modificamos el esquema de traducción dirigido por la sintaxis de Minicomp añadiendo esto:

```
<Termino> -> "#(" @nlinea = mc_al.linea()@ <Expresion> @subarboles = [Expresion1.arb]@
             ( "," <Expresion> @subarboles.append(Expresion2.arb)@ )*
             ")" @Termino.arb = AST.NodoContarEnExpresiones(subarboles, nlinea)@ ;
```

En este ejercicio se pide escribir, únicamente, la función de generación de código de `NodoContarEnExpresiones`.

Prueba 2 (0.50 puntos) Supongamos ahora que queremos modificar Minicomp para que el lenguaje de entrada acepte una nueva operación que permita contar la cantidad de veces que aparece un valor entre los elementos de un vector, empleando la sintaxis

`#[` *expresión* `,` *vector* `]`

donde los símbolos `#[` de la apertura van unidos sin espacio entre ellos, *expresión* es una expresión de tipo entero y *vector* es un identificador, posiblemente seguido de expresiones entre corchetes. El tipo de *vector* es vectorial con tipo base entero. La evaluación de la expresión se hace antes que el acceso al vector, y da el valor que se desea contar. Se garantiza que los posibles efectos secundarios de evaluar la expresión y del acceso al vector se producen una sola vez, y que la longitud del código generado (medida en instrucciones) es independiente de la talla del vector. Es responsabilidad del usuario que las componentes del vector hayan sido inicializadas previamente. La nueva construcción es válida en cualquier contexto en que sea válida una expresión de tipo entero. Por ejemplo, el siguiente programa escribe 22:

```
globales
   v: vector[3] de entero;
   m: vector[4] de vector[2] de entero;
fin
secuencia
   v[0] := 2 + 3;
   v[1] := 4;
   v[2] := 5;
   m[2][0] := 4 + 1;
   m[2][1] := 5;
   escribe #[2 + 3, v] + 10 * #[5, m[1+1]];
fin
```

En este ejercicio se pide (i) indicar las modificaciones que harías en el esquema de traducción para analizar esa nueva construcción y representarla con un objeto de la clase `NodoContarEnVector`; y (ii) escribir la función de generación de código de dicha clase.