UNIVERSITAT JAUME·I

Ingeniería Informática
Procesadores de lenguaje
Examen (13 de diciembre de 2010)

PREGUNTA 1 (5 PUNTOS)

Explica cómo debería modificarse la versión 9 de `minicomp` del curso 2009/10 de modo que aceptara las siguientes extensiones, que son independientes entre sí.

Las descripciones de modificaciones del nivel léxico deben explicitar, si las hay, las nuevas expresiones regulares y los atributos asociados a posibles nuevas categorías, **no utilices categorías sin nombre con metacomp para las categorías que crees**; las de modificaciones del nivel sintáctico o semántico deben aclarar las reglas, acciones semánticas o métodos de comprobación afectados; las de modificaciones de la generación de código deben dejar claro qué instrucciones de máquina virtual se emitirían con la modificación. Por ejemplo: sería aceptable una frase como "se comprobaría que el hijo izquierdo es de tipo entero" (es trivial transformar la comprobación en un test), pero no una como "se generaría código para sumar los dos números" (no sabemos si se generaría una instrucción o varias, cuál o cuáles serían, dónde están los números —¿en registros, en memoria, en la pila?—, etc.).

Explicita cualquier asunción que hagas acerca del enunciado propuesto.

E1. Operadores de desplazamiento aritmético (3 puntos)

Esta extensión introduce dos nuevos operadores, `<<` y `>>`, que permiten realizar desplazamientos aritméticos a izquierda y derecha, respectivamente. Ambos son binarios, infijos, asociativos por la derecha y tienen la misma prioridad, que es menor que la del operador de disyunción. Los dos operandos que reciben deben tener tipo entero, que también es el tipo de su resultado. Su evaluación comporta la evaluación de sus operandos una sola vez y el desplazamiento aritmético del operando izquierdo tantas posiciones como indique el operando derecho en la dirección correspondiente (izquierda para `<<` y derecha para `>>`). Recuerda que el desplazamiento aritmético a la izquierda una posición equivale a una multiplicación por dos. Si el desplazamiento es a la derecha, equivale a una división por dos. En caso de que el número de posiciones del desplazamiento sea menor o igual que cero, se devuelve como resultado el operando izquierdo.

Por ejemplo, el siguiente fragmento

```
para i = 0 hasta 8 hacer
  escribe 2-1 << i, " ", 255+1 >> i; nl;
fin
```

escribe:

```
1 256
2 128
4 64
8 32
16 16
32 8
64 4
128 2
256 1
```

E2. Operador de comparación de cadenas con vectores (2 puntos)

Esta extensión permite utilizar la nueva palabra reservada `representa` como un nuevo operador lógico que permite saber si un vector de enteros contiene los códigos ASCII correspondientes a los caracteres de una cadena. Más concretamente, la expresión

`representa(`*`vector`*`, `*`cadena`*`)`

donde *`vector`* es una expresión de tipo vector de enteros y *`cadena`* una expresión de tipo cadena, es cierta si y sólo si se cumple que:

- La talla de *`vector`* es igual a la longitud de *`cadena`*.
- Para todo i entre uno y la longitud de la cadena, *`v`*`[`$i-1$`]` es igual al código ASCII del i-ésimo carácter de *`cadena`*.

Por ejemplo, el siguiente programa

```
globales
   vc: vector[2] de cadena;
   ve: vector[5] de entero;
   i: entero;
fin

secuencia
  ve[0] = 104; // h
  ve[1] = 111; // o
  ve[2] = 108; // l
  ve[3] = 97;  // a
  ve[4] = 33;  // !

  vc[0] = "hola";
  vc[1] = "hola!";
  para i = 0 hasta 1 hacer
    si representa(ve, vc[i]) entonces
      escribe vc[i]; nl;
    fin
  fin
fin
```

escribe `hola!`.

Al usar este operador, la cantidad de instrucciones del código generado debe ser independiente de la talla del vector y la cadena. También se garantiza que los efectos secundarios de evaluar las expresiones suceden una sola vez.

Pregunta 2 (1.5 puntos)

Escribe tanto un AFD como una expresión regular para modelar cada uno de los siguientes lenguajes:

- El de los identificadores que no contienen más de tres aes.
- El de los identificadores que no contienen más de tres aes seguidas.

En todos los casos, considera los identificadores como secuencias de una o más letras minúsculas.

Pregunta 3 (2 puntos)

La siguiente gramática, G_1, genera el lenguaje de todas las expresiones que se pueden formar utilizando correctamente números enteros y dos posibles operadores, además de paréntesis: (i) resta, que es binario, infijo y asociativo por la izquierda; y (ii) cambio de signo, que es unario, prefijo y más prioritario que la resta.

$$
\begin{array}{lcl}
\langle \text{Expresión} \rangle & \rightarrow & \langle \text{Expresión} \rangle \ \textbf{menos} \ \langle \text{Expresión} \rangle \\
\langle \text{Expresión} \rangle & \rightarrow & \textbf{menos} \ \langle \text{Expresión} \rangle \\
\langle \text{Expresión} \rangle & \rightarrow & \textbf{abreParéntesis} \ \langle \text{Expresión} \rangle \ \textbf{cierraParéntesis} \\
\langle \text{Expresión} \rangle & \rightarrow & \textbf{entero}
\end{array}
$$

La siguiente gramática, G_2, es un intento de modelar, con una gramática RLL(1), el mismo lenguaje, teniendo en cuenta las prioridades y asociatividades de los operadores:

$$
\begin{array}{lcl}
\langle \text{Expresión} \rangle & \rightarrow & \langle \text{Término} \rangle \ (\textbf{menos} \ \langle \text{Expresión} \rangle)^* \\
\langle \text{Término} \rangle & \rightarrow & \textbf{menos} \ \langle \text{Término} \rangle \\
\langle \text{Término} \rangle & \rightarrow & \textbf{abreParéntesis} \ \langle \text{Término} \rangle \ \textbf{cierraParéntesis} \\
\langle \text{Término} \rangle & \rightarrow & \textbf{entero}
\end{array}
$$

Responde a las siguientes preguntas:

a) ¿Es G_2 ambigua? ¿Por qué?

b) ¿Es G_2 RLL(1)? ¿Por qué?

c) ¿Genera G_2 el mismo lenguaje que G_1? ¿Por qué?

d) Corrige G_2, si hace falta, para que sea RLL(1) y genere el mismo lenguaje que G_1, teniendo en cuenta las prioridades y asociatividades de los operadores. Demuestra que el resultado es RLL(1).

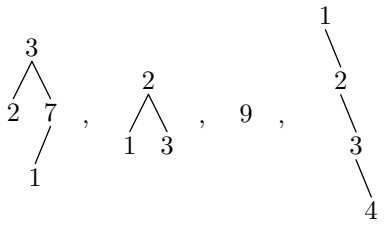
Si lo deseas, en tu respuesta puedes utilizar las iniciales de los símbolos de la gramática.

Pregunta 4 (1,5 puntos)

Considera la siguiente gramática, que modela un bosque formado por una secuencia de árboles binarios separados por comas:

$$
\begin{array}{rcl}
\langle \text{Bosque} \rangle & \rightarrow & \langle \text{Arbin} \rangle \langle \text{MásArbin} \rangle \\
\langle \text{MásArbin} \rangle & \rightarrow & \textbf{coma}\ \langle \text{Arbin} \rangle \langle \text{MásArbin} \rangle \mid \lambda \\
\langle \text{Arbin} \rangle & \rightarrow & \textbf{abreParéntesis}\ (\textbf{entero}\ (\langle \text{Arbin} \rangle \langle \text{Arbin} \rangle)?)?\ \textbf{cierraParéntesis}
\end{array}
$$

Por ejemplo, el bosque



se puede representar mediante la cadena

```
(3(2()())(7(1)())), (2(1)(3)), (9), (1()(2()(3()(4))))
```

Añade acciones semánticas a la gramática anterior, sin modificarla, para que el esquema de traducción resultante escriba, al finalizar un análisis RLL(1) de la entrada, el mensaje "todos sí" si todos los árboles del bosque son árboles binarios de búsqueda y "alguno no" si alguno no lo es.

Sólo puedes utilizar atributos de tipo lógico y entero (y no, por ejemplo, de tipo lista). Además, no puedes utilizar ningún objeto global ni tampoco puedes utilizar como heredados atributos sintetizados y viceversa. Asume que los componentes de tipo **entero** tienen un atributo, v, que contiene el valor del entero (y que puede ser negativo).

Indica para cada atributo si es heredado o sintetizado y qué representa.

Nota: consideramos que un árbol binario es un árbol binario de búsqueda cuando para cualquier nodo x del árbol todos los nodos de su subárbol izquierdo son menores que x y todos los nodos de su subárbol derecho son mayores que x. En nuestro ejemplo, el primer árbol no lo es y los tres siguientes sí. Por lo tanto, se emitiría el mensaje "alguno no".

Duración del examen: 4 horas