UNIVERSITAT JAUME·I

Ingeniería Informática
Procesadores de lenguaje
Examen (11 de diciembre de 2007)

PREGUNTA 1 (5 PUNTOS)

Explica cómo debería modificarse `minicomp` (en su versión para Stan o para Rossi, pero sin ninguna de las extensiones planteadas en las prácticas) de modo que aceptara las extensiones que se presentan a continuación. Las extensiones son independientes entre sí; no hace falta que consideres sus posibles interacciones.

Procura que tu descripción sea clara, escueta y precisa. Para ello, puede facilitarte la exposición una estructura que siga las distintas etapas del compilador.

Puedes optar por descripciones algorítmicas o en lenguaje natural para lograr una mayor sencillez en la explicación, pero dejando claro qué se modificaría y cómo. Como mínimo: las descripciones de modificaciones del nivel léxico deben explicitar, si las hay, las nuevas expresiones regulares y los atributos asociados a posibles nuevas categorías; las de modificaciones del nivel sintáctico o semántico deben aclarar las reglas, acciones semánticas y/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. Inversión de vectores (3 puntos)

Esta extensión introduce una nueva sentencia que permite invertir un vector. Para ello, se emplea la sintaxis

```
invierte (vector);
```

donde `invierte` es una nueva palabra reservada y *`vector`* un identificador, posiblemente seguido de expresiones entre corchetes. El tipo de *`vector`* debe ser vectorial con tipo base simple. El resultado de la ejecución es que el primer elemento se intercambia con el último, el segundo con el penúltimo y así sucesivamente. Por ejemplo, dadas las declaraciones

```
v: vector[4] de cadena;
matriz: vector [5] de vector [5] de entero;
```

las sentencias

```
invierte(v); invierte(matriz[2]);
```

invertirían todos los elementos de `v` y los de la tercera fila de `matriz`.

Nota: los posibles efectos secundarios del acceso a *`vector`* se producen sólo una vez.

E2. Polinomios (2 puntos)

Mediante esta extensión, el programador puede evaluar polinomios de una variable en un punto. Para ello, se representa el polinomio escribiendo los coeficientes en una lista encerrada entre corchetes seguida por el valor de x entre paréntesis. Por ejemplo, para evaluar x^2+3x+2 en $x=4$, se escribiría `[1,3,2](4)`. Tanto los coeficientes como el valor de x deben ser expresiones de tipo entero por lo que el resultado también lo será.

Algunos ejemplos de expresiones con polinomios junto con su valor son:

Expresión	Valor
`2*[3,2](4)`	28
`[1,1+[1+2](3),1](4+3)/2`	39
`15+[1,1+1,1+1+1](2*2)`	42

Notas:

- Se garantiza que los posibles efectos secundarios de la evaluación de x suceden sólo una vez.
- Se puede evaluar fácilmente un polinomio p mediante el algoritmo siguiente:

  $r := 0$;
  para $i := n$ **hasta** 0 **salto** -1 **hacer**
  $\quad r := r \cdot x + p_i$;
  devolver r;

  donde p_i es el coeficiente del término de grado i.

Pregunta 2 (1,5 puntos)

En un lenguaje de programación ficticio, los comentarios comienzan con la secuencia de tres caracteres `<<<` y terminan al llegar a un salto de linea o al aparecer la secuencia de tres caracteres `>>>`, lo que suceda primero. Modela, mediante una expresión regular, el conjunto de estos comentarios. Debes tener en cuenta que el delimitador `>>>` formaría parte del comentario, pero el salto de linea no formaría parte del comentario.

Pregunta 3 (1,5 puntos)

Demuestra que la siguiente gramática es ambigua:

$$
\begin{array}{lcl}
\langle \mathsf{S} \rangle & \rightarrow & \langle \mathsf{A} \rangle \langle \mathsf{B} \rangle \,|\, \langle \mathsf{C} \rangle \\
\langle \mathsf{A} \rangle & \rightarrow & \mathbf{a} \langle \mathsf{A} \rangle \mathbf{b} \,|\, \mathbf{ab} \\
\langle \mathsf{B} \rangle & \rightarrow & \mathbf{c} \langle \mathsf{B} \rangle \mathbf{d} \,|\, \mathbf{cd} \\
\langle \mathsf{C} \rangle & \rightarrow & \mathbf{a} \langle \mathsf{C} \rangle \mathbf{d} \,|\, \mathbf{a} \langle \mathsf{D} \rangle \mathbf{d} \\
\langle \mathsf{D} \rangle & \rightarrow & \mathbf{b} \langle \mathsf{D} \rangle \mathbf{c} \,|\, \mathbf{bc}
\end{array}
$$

Pregunta 4 (2 puntos)

Sea G la siguiente gramática, que genera expresiones formadas por enteros y operadores de suma:

$$
\begin{array}{rcl}
\langle \mathsf{S} \rangle & \rightarrow & \langle \mathsf{Expr} \rangle \\
\langle \mathsf{Expr} \rangle & \rightarrow & \langle \mathsf{Expr} \rangle \langle \mathsf{Expr} \rangle \,|\, \mathbf{díg} \,|\, \langle \mathsf{Suma} \rangle \\
\langle \mathsf{Suma} \rangle & \rightarrow & \mathbf{díg+díg} \,|\, \mathbf{díg\ díg}
\end{array}
$$

donde **díg** es un terminal que representa un dígito del uno al nueve con valor **díg**.v. Supongamos que la regla inicial tiene la acción semántica siguiente:

$$\langle \mathsf{S} \rangle \rightarrow \langle \mathsf{Expr} \rangle \ \{\textbf{escribe}(\text{“La suma es ”}, \langle \mathsf{Expr} \rangle.\text{suma});\}$$

con el objetivo de escribir cuál es la suma de la expresión. Por ejemplo, para la expresión `15+27` la suma es 42.

Añade, *al final de cada una de las restantes reglas*, las acciones semánticas necesarias para que se calcule el valor del atributo suma de ⟨Expr⟩. Puedes utilizar los atributos adicionales que consideres necesarios, pero ninguna variable global. Además, los atributos que añadas deben ser de tipo entero o lógico.

Nota: puede serte útil el operador $\oplus$ que, dados dos enteros, los concatena como si fueran cadenas. Por ejemplo: $12 \oplus 84 = 1284$.

Duración del examen: 4 horas