UNIVERSITAT JAUME·I

Ingeniería Informática

Procesadores de lenguaje

Examen (19 de diciembre de 2008)

PREGUNTA 1 (5 PUNTOS)

Explica cómo debería modificarse la versión 7 de `minicomp` (la que incluye las modificaciones hasta "promociones implícitas de entero a real", inclusive) 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. Operadores de incremento y decremento (3 puntos)

Esta extensión introduce cuatro operadores unarios: preincremento (`++` en forma prefija), postincremento (`++` en forma postfija), predecremento (`--` en forma prefija) y postdecremento (`--` en forma postfija), de modo que los postfijos son más prioritarios que los prefijos y todos ellos son más prioritarios que el resto de operadores. El operando debe ser de tipo entero y tener *valor-i*. Además, los posibles efectos secundarios de su evaluación se producen sólo una vez. Así, en `escribe v[llama f()]++;`, donde `v` es un vector de enteros, la función `f` es llamada una sola vez.

Por ejemplo, el siguiente programa:

```
globales
  a, b: entero;
fin
secuencia
  a = 10;
  b = ++a;
  escribe a; escribe " "; escribe b; escribe " ";
  a = 20;
  b = a++;
  escribe a; escribe " "; escribe b; escribe " ";
fin
```

escribiría en pantalla `11 11 21 20`.

En cambio, si añadimos al programa anterior la siguiente línea:

```
b = ++a--;
```

el compilador debería detectar un error semántico, puesto que el operando del preincremento no tiene *valor-i*.

E2. Operador barra (2 puntos)

Esta extensión introduce el operador unario barra $|\cdot|$. El operando debe ser una expresión de tipo entero, real o cadena, encerrada entre las dos barras verticales. Si el operando es de tipo entero o real, el resultado es su valor absoluto; si es de tipo cadena, su longitud. Por ejemplo, la sentencia `escribe |2-|"cadena"||;` escribiría `4`.

El operador garantiza que la expresión se evalúa exactamente una vez.

Pregunta 2 (2 puntos)

Escribe tanto un AFD como una expresión regular para modelar el lenguaje de las cadenas de dígitos que empiezan por `111`, por `12` o por `21`, y terminan con la primera de esas tres subsecuencias que aparezca tras la inicial.

Supón que el alfabeto está formado sólo por dígitos.

Pregunta 3 (1,5 puntos)

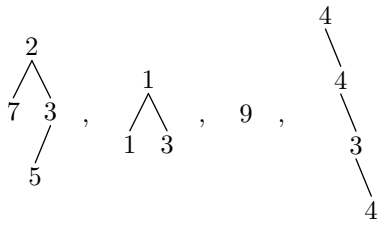
Sea G una gramática LL(1) y $\langle \mathsf{A} \rangle$ un no terminal para el que existe una única regla $\langle \mathsf{A} \rangle \to \alpha$. Llamemos $G_{\langle \mathsf{A} \rangle}$ a la gramática en la que todas las apariciones de $\langle \mathsf{A} \rangle$ en partes derechas de las reglas de G han sido sustituidas por α. Demuestra que $G_{\langle \mathsf{A} \rangle}$ es LL(1).

Pregunta 4 (1,5 puntos)

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

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

Por ejemplo, el bosque



se puede representar mediante la cadena

```
(2(7()())(3(5)())), (1(1)(3)), (9), (4()(4()(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 tienen orden de montículo y "alguno no" si alguno no lo tiene.

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 tiene orden de montículo si todos los nodos tienen un valor menor o igual que cualquiera de sus hijos, si estos existen. En nuestro ejemplo, los tres primeros árboles lo cumplen y el último no. Por lo tanto, se emitiría el mensaje "alguno no".

Duración del examen: 4 horas