UNIVERSITAT JAUME·I

Ingeniería Informática

Procesadores de lenguaje

Examen de teoría (15 de septiembre de 2006)

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 la página de la asignatura) 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.

En tu descripción, procura ser claro, escueto y preciso. 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 enunciado propuesto.

Recorrido de vectores (3 puntos)

Esta modificación introduce una nueva estructura de control que permite recorrer vectores fácilmente. Para ello, se emplea la sintaxis

```
recorre vector con variable hasta último haciendo
  sentencia
```

donde *vector* y *variable* son identificadores, posiblemente seguidos de una o más expresiones entre corchetes, y *último* es una expresión de tipo entero. El tipo de *variable* debe ser el mismo que el tipo base de *vector*.

La ejecución de la estructura supone la ejecución de *sentencia* una vez por cada uno de los elementos de *vector* con índices entre cero y *último*, ambos inclusive, y con el valor correspondiente almacenado en *variable*. Por ejemplo, suponiendo la declaración `v: vector [5] de vector [50] de entero;` el fragmento

```
recorre v[2] con v[1][0] hasta 3+2 haciendo
  escribe(v[1][0]);
```

escribiría los valores de `v[2][0]` a `v[2][5]`.

Notas:

- Los posibles efectos secundarios de evaluar *vector*, *variable* y *último* se producen sólo una vez.
- Está indefinido el comportamiento del bucle si *variable* forma parte de *vector*.
- Está indefinido el valor que tendrá *variable* al salir del bucle.
- Si el valor de *último* no fuera menor que la longitud del vector, se ejecutaría *sentencia* una vez por cada elemento del vector; si fuera negativo, no se ejecutaría *sentencia* ninguna vez.
- El código generado no puede crecer exponencialmente con el tamaño del programa fuente.

Automodificadores (2 puntos)

Esta modificación introduce cinco nuevos operadores binarios: `(+)`, `(-)`, `(*)`, `(/)` y `(%)`, a los que llamaremos automodificadores. Si *op* es un operador binario (`+`, `-`, `*`, `/` o `%`), el correspondiente automodificador, `(`*op*`)`, tiene su misma prioridad y asociatividad. Al ejecutarse, devolverá el mismo resultado, pero con la particularidad de que el valor devuelto se almacenará en su operando izquierdo, que debe ser un valor-i.

Por ejemplo, el fragmento

```
a:=1; v[1]:= 2;
escribe(a(+)v[1](*)3); escribe(", ")
escribe(a); escribe(", "); escribe(v[1]);
```

escribiría en pantalla `7, 7, 6`.

Notas:

- En la parte izquierda de un automodificador se permite la utilización de paréntesis.
- Los posibles efectos secundarios de la evaluación de los operandos de los automodificadores suceden una sola vez y en orden de izquierda a derecha.

Además de las modificaciones, escribe el código que se generaría para la sentencia `escribe(a(+)v[1](*)3);`. Da a las variables las direcciones que consideres oportunas.

Pregunta 2 (1,5 puntos)

Dada una expresión regular r definimos $\text{ne}(r)$ como el número de estados del AFD construido mediante el método de los ítems. Dadas las expresiones regulares r y s, demuestra la verdad o falsedad de las siguientes proposiciones:

- Si $\text{ne}(r) \leq \text{ne}(s)$, entonces $L(r) \subseteq L(s)$.
- Si $L(r) \subseteq L(s)$, entonces $\text{ne}(r) \leq \text{ne}(s)$.
- $\text{ne}(r) \leq \text{ne}(r \mid s)$.

Pregunta 3 (2 puntos)

Supongamos que tenemos la siguiente gramática para las expresiones de un lenguaje de programación:

$$\begin{array}{lcl} \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle \ \texttt{+} \ \langle \mathsf{T} \rangle \mid \langle \mathsf{T} \rangle \\ \langle \mathsf{T} \rangle & \rightarrow & \langle \mathsf{T} \rangle \ \texttt{*} \ \langle \mathsf{F} \rangle \mid \langle \mathsf{F} \rangle \\ \langle \mathsf{F} \rangle & \rightarrow & \texttt{entero} \mid \texttt{cadena} \mid \texttt{(} \langle \mathsf{E} \rangle \texttt{)} \end{array}$$

Como ves, son expresiones sobre enteros y cadenas. Las reglas semánticas permiten la suma y el producto de enteros, la concatenación de cadenas (mediante `+`) y la repetición de cadenas un número de veces (mediante `*` aplicado a una cadena y un entero, en cualquier orden).

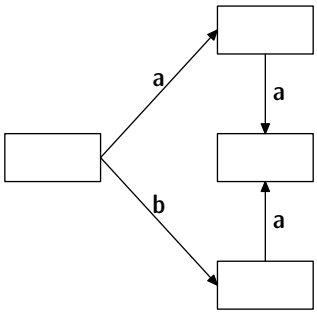
Añade *al final de cada regla*, las acciones necesarias para sintetizar en el símbolo inicial $\langle \mathsf{E} \rangle$ los siguientes atributos:

- *tipo*, que contiene el tipo de la expresión (entero, cadena o error),
- *mse*, que contiene la longitud del producto de literales enteros más largo que multiplica *directamente* (sin paréntesis) un literal de cadena. Algunos ejemplos de expresiones con los correspondientes valores del atributo son:

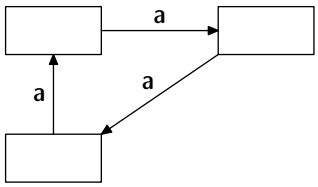
Expresión	*mse*
`entero*cadena*entero*entero`	2
`entero*cadena*(entero*entero)`	1
`(entero*cadena)*entero*entero`	1
`cadena*(entero+entero)*entero`	0

Pregunta 4 (1,5 puntos)

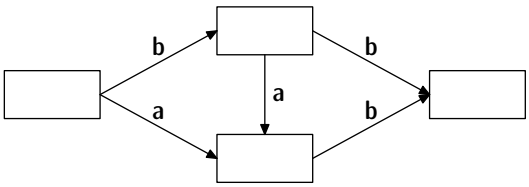
A continuación hay tres porciones de otros tantos autómatas de prefijos viables. Por cada uno, escribe una gramática SLR cuyo autómata contenga la parte mostrada o demuestra que ello no es posible.



(a)



(b)



(c)

Duración del examen: 4 horas
¡Buena suerte!