E79 Procesadores de lenguaje

Examen de teoría (31 de mayo de 2003)

PREGUNTA 1 (6 PUNTOS)

A continuación, se presentan tres posibles extensiones del lenguaje MM3. Elige dos de ellas y explica claramente qué modificaciones se tendrían que hacer en un compilador de MM3 a Stan para que las aceptara. Las modificaciones son independientes entre sí; no hace falta que consideres sus posibles interacciones.

En tu descripción de las modificaciones, procura ser claro, escueto y preciso. En particular, no es necesario que describas partes del compilador que no estén afectadas por las modificaciones. 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 compilador o del enunciado propuesto.

Variables autoincrementadas

Esta extensión crea un nuevo tipo de variables, las *autoincrementadas*. Estas son variables enteras que tienen la particularidad de incrementar su valor cada vez que son leídas. Es decir, cuando aparecen en una expresión devuelven su valor actual y después lo incrementan. El valor del incremento se puede fijar en su declaración y por defecto es uno. El comportamiento de estas variables en la parte izquierda de las asignaciones es el habitual.

Para declarar la variable como autoincrementada, se escribe en su declaración el componente léxico `++` tras su identificador. Si se quiere fijar un incremento distinto de la unidad, se escribe el correspondiente valor entre paréntesis tras el `++`. El valor del incremento sigue la misma sintaxis que el valor que se asocia a las constantes. Así, los siguientes son ejemplos de variables autoincrementadas:

```
entero a++, b++(3);
entero c++(-d); *** d es constante ***
```

En cuanto a su uso, con las declaraciones anteriores, la salida del siguiente fragmento de código:

```
a<- 1; b<- 1;
repite 5 veces:
  salida <- "(" <- a <- "," <- b <- ") ";
fin
salida <- "$n";
```

sería:

```
(1,1) (2,4) (3,7) (4,10) (5,13)
```

Variables locales persistentes

Esta extensión permite crear variables locales a las subrutinas que retienen su valor entre llamadas. Para poder declararlas se introduce en MM3 la palabra reservada `persistente`, con las reglas habituales para palabras reservadas. Cuando se antepone esta palabra al tipo en una declaración de variables locales, se indica que el valor de esas variables se conservará entre las distintas ejecuciones. El valor inicial de las variables se fija siguiendo el correspondiente identificador con un símbolo de asignación y un valor constante según las reglas de declaración de constantes. En caso de no fijarse valor alguno, la variable se inicializa a cero. Por ejemplos, supongamos que las siguientes declaraciones se encuentran en una función `f`:

```
persistente entero a, b<- 4;
persistente real c<- 3.14;
```

La primera vez que se ejecute `f`, la variable `a` tendrá el valor 0, `b` valdrá 4 y c 3,14. En las siguientes llamadas a `f`, estas variables conservarán el valor de la llamada anterior. Otro ejemplo sería la siguiente definición de la función `contador`:

```
subrutina contador () devuelve entero:
  persistente entero c;
  c<- c+1;
  devuelve c;
fin
```

La primera llamada a esta función devolverá 1, la segunda 2 y así sucesivamente.

No está definido qué sucede con una variable persistente en una invocación de una subrutina recursiva cuando en otra invocación se le asigna un valor.

Saltos incondicionales

Esta extensión introduce la nueva instrucción `salto` que permite realizar saltos incondicionales a etiquetas definidas en el programa. Las etiquetas se pueden definir delante de cualquier sentencia mediante un número entero estrictamente positivo seguido del carácter dos puntos (`:`) sin ningún espacio intermedio. La instrucción `salto` tiene la forma

`salto` *expresión* ;

Su ejecución provoca la evaluación de la expresión (que tiene que ser entera) y el salto a la etiqueta cuyo entero asociado coincida con el resultado de la expresión. El siguiente ejemplo de uso muestra cómo se podría implementar un menú sencillo:

```
salida <- "¿Que quieres hacer? 1.- Leer, 2.- Escribir?";
entrada -> opcion;
salta opcion;
...
1: *** Código de lectura ***
...
salta 3;
2: *** Código de escritura ***
...
3: *** Fin ***
```

Está indefinido qué pasa si se salta desde una función al cuerpo de otra, de una función al programa principal o de este al cuerpo de una función. Así mismo, está indefinido qué sucede si no existe la etiqueta destino del salto o si el salto o su destino están en el cuerpo de un bucle.

Pregunta 2 (2 puntos)

Rellena el siguiente cuadro, indicando, con `SI` o `NO`, si la cadena es aceptada por la correspondiente expresión regular:

	λ	`z?a`	`az*`	`??`	`b-c?`	`(a)`	`(b`	`)`	`\)`	`\\)`
`[^az*]?`										
`([^a]?[^z]?)*`										
`([^az]?)*`										
`\([^az]?\)*`										
`[^(az)*]?`										

Pregunta 3 (2 puntos)

Sea $G_n = (N, \Sigma, P, \langle \mathsf{A}_1 \rangle)$ una gramática incontextual con:

- $N = \{\langle \mathsf{A}_1 \rangle, \ldots, \langle \mathsf{A}_n \rangle\}$,
- $\Sigma = \{a_1, \ldots, a_n\}$

y las reglas:

$$\begin{array}{lcl} \langle \mathsf{A}_1 \rangle & \rightarrow & \langle \mathsf{A}_2 \rangle \langle \mathsf{A}_2 \rangle | a_1 \\ \langle \mathsf{A}_2 \rangle & \rightarrow & \langle \mathsf{A}_3 \rangle \langle \mathsf{A}_3 \rangle | a_2 \\ \ldots & & \\ \langle \mathsf{A}_n \rangle & \rightarrow & a_n \end{array}$$

Se pide:

- Calcular el número de estados del autómata de prefijos viables de G_n y determinar si la gramática es SLR.
- Ídem en el caso de que existan exactamente un i y un j tales $a_i = a_j$ e $i < j$.