E79 Procesadores de lenguaje

Examen de teoría (9 de septiembre 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.

Extremos de los vectores

Esta modificación introduce dos nuevos operadores unarios: `min` y `max` (`min` y `max` son nuevas palabras reservadas en MM3). Estos operadores reciben como único parámetro un identificador de vector (entre paréntesis) y devuelven el valor mínimo (`min`) o máximo (`max`) almacenado en él. El tipo del resultado es el tipo base del vector.

Un ejemplo de uso sería:

```
...
lee_vector(); *** Almacena los valores en v ***
salida <- "El mínimo de los números es " <- min(v);
salida <- " y el máximo " <- max(v) <- "$n";
...
```

Nota: el tamaño del código generado no puede crecer exponencialmente con la talla del programa fuente.

Operadores postfijos definidos por el usuario

Mediante esta extensión se permite al usuario definir nuevos operadores postfijos. Estos operadores se componen de un carácter circunflejo (`^`) seguido de, al menos, una letra, dígito o subrayado (algunos operadores válidos serían: `^n`, `^3m`, `^2_a` y `^_max`). Su prioridad es mayor que la de los operadores unarios prefijos ya definidos en la gramática (esto es, `-x^2` se entiende como `-(x^2)` y no como `(-x)^2`).

Para definirlo, se utiliza la misma sintaxis que para el resto de operadores definidos por el usuario con la única particularidad de que en la expresión no se puede utilizar la palabra reservada `der`, pero sí la palabra `izq`, que contiene el único parámetro del operador. Así, podemos definir un operador para calcular cuadrados y otro para calcular cuartas potencias de la siguiente manera:

```
operador entero ^2: izq*izq;
operador entero ^4: izq^2^2;
```

Luego podemos emplearlos en el código normalmente:

```
entero x;
x<- 10;
salida<- -x^2+x^4;
```

escribiría por pantalla 9900.

Bucles con incrementos distintos de la unidad

Esta extensión permite que los bucles de tipo `REPITE` puedan incrementar la variable `CONTADOR` en más de una unidad por vuelta del bucle. Para ello, el programador puede escribir opcionalmente, entre `VECES` y los dos puntos, la nueva palabra reservada `SALTANDO` y una expresión. La expresión será de tipo entero y se evaluará una sola vez al comienzo del bucle. El resultado (que puede ser negativo o cero) será el que se sume a `CONTADOR` en cada vuelta.

Así, el bucle

```
REPITE 10 VECES SALTANDO 3:
  salida<- CONTADOR;
FIN
```

escribirá por pantalla los números `1`, `4`, `7`, `10`, `13`, `16`, `19`, `22`, `25` y `28`, mientras que el bucle

```
REPITE 10 VECES SALTANDO -3:
  salida<- CONTADOR;
FIN
```

escribirá por pantalla los números `1`, `-2`, `-5`, `-8`, `-11`, `-14`, `-17`, `-20`, `-23` y `-26`. Si el resultado de la expresión del salto es cero, el bucle se ejecutará el número de veces indicado, pero el valor de `CONTADOR` será siempre uno.

Pregunta 2 (1 punto)

Las siguientes expresiones regulares son intentos de modelar los comentarios de un determinado lenguaje de programación. En este lenguaje, los comentarios comienzan con la secuencia `*->` y terminan en `<-*`, sin ninguna aparición de `<-*` en su interior. Por cada expresión, di si es o no correcta y, si no lo es, escribe una cadena para la que falle.

1. `\*->[^<–*]*<-\*`
2. `\*->([^<]|[^-]|[^*])*<-\*`
3. `\*->([^<]|<[^-]|<-[^*])*<-\*`
4. `\*->([^<]|<+[^-<]|<+[-<]+[^-*<])*<-\*`

Pregunta 3 (3 puntos)

Escribe una gramática SLR con símbolo inicial $\langle A \rangle$ y tal que uno de los estados de su autómata de prefijos viables contenga, entre otros, los ítems: $\langle A \rangle \to \langle B \rangle a \cdot \langle C \rangle$, $\langle B \rangle \to a \cdot \langle E \rangle$ y $\langle C \rangle \to a \cdot \langle F \rangle$. Escribe el autómata completo de tu gramática y demuestra que no tiene conflictos SLR.