UNIVERSITAT JAUME·I

5º Ingeniería Informática

E79 Procesadores de lenguaje

Examen de teoría (13 de diciembre 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.

Aplicación de funciones a vectores

Esta modificación extiende el significado de la sintaxis `f[v]` para cubrir el caso en que `f` es una función y `v` un vector. El efecto de esta expresión es que se calcula el valor de `f` tomando como entrada cada uno de los valores del vector y se almacena el resultado en la posición correspondiente. Como resultado de la expresión, se devuelve el último valor almacenado. Por ejemplo:

```
vector entero [1..5] ve;
repite 5 veces:
  ve[contador]<- contador;
fin
salida <- dobla[ve]+1;
subrutina dobla (entero n) devuelve entero:
  devuelve 2*n;
fin
```

escribiría `11` por pantalla y dejaría en el vector los elementos `2`, `4`, `6`, `8` y `10`.

La función que se aplica al vector debe tener exactamente un parámetro, de tipo igual o más general que el tipo base del vector y debe devolver un resultado de tipo igual o menos general que el tipo base del vector.

Nota: no se permite que la longitud del código generado crezca exponencialmente con la talla del programa de entrada.

Operador potencia

Esta modificación añade un nuevo operador a MM3, el operador potencia, representado mediante `**`. Este operador es asociativo por la derecha y es el operador binario de más prioridad, aunque tiene menos prioridad que los operadores unarios. Acepta en la base expresiones de tipo entero o real, mientras que en el exponente sólo acepta expresiones de tipo entero. El tipo del resultado es el tipo de la base. El cálculo se hace de la manera natural: si el exponente es positivo, se multiplica la base el número de veces indicado por el exponente; si el exponente es cero, el resultado es uno; y si el exponente es negativo el resultado es el de dividir uno entre la base elevada al exponente cambiado de signo.

Por ejemplo:

```
salida <- 2**3**2 <- ", " <-  2**0 <- ", " <- 2**-2 <- ", " <- 2.0**-2;
```

escribiría en pantalla `512, 1, 0, 0.25`.

Pista: la implementación puede ser más fácil si codificas parte del código utilizando el propio MM3.

Argumentos con nombre

Con esta modificación se extiende la sintaxis de las llamadas a función para permitir poner, delante de las expresiones correspondientes a los argumentos, el nombre del parámetro al que se refieren. Para especificar a qué parámetro corresponde el argumento, se escribe el nombre del parámetro, un componente asignación (`<-`) y la expresión. En una misma llamada se permite mezclar argumentos con y sin nombre, siempre que todos los argumentos que sigan a uno con nombre tengan nombre. Aquellos argumentos que preceden a los argumentos con nombre se asignan a los parámetros de la forma habitual (por posición).

Por ejemplo, dada la función con perfil

```
subrutina f(entero p, real q) devuelve real:
```

Las siguientes son posibles llamadas y algunos ejemplos de errores:

```
r<- f(1, 2);
r<- f(1, q<- 1.2);
r<- f(p<- 1, q<- 1.2);
r<- f(q<- 1.2, p<- 1);
r<- f(q<- 1.2, p<- 1.0); *** Error: p debe ser entero. ***
r<- f(q<- 1.2, 1); *** Error: argumento sin nombre tras argumento con nombre. ***
r<- f(q<- 1.2); *** Error: faltan parámetros. ***
r<- f(p<- 1, q<- 1.2, p<-2); *** Error: p especificado dos veces. ***
```

Pregunta 2 (2 puntos)

Describe brevemente cómo organizarías la parte del analizador léxico que controla la indentación si tuvieras que escribir un intérprete Python. Observa en el siguiente ejemplo los componentes emitidos para cada una de las líneas:

`for i in l:`	**for id in id ptos nl**
`  for j in l:`	**dentro for id in id ptos nl**
`    for k in l:`	**dentro for id in id ptos nl**
`     print k`	**dentro print id nl**
`  print i`	**fuera fuera print id nl**
$_{f}^{e}$	**fuera eof**

Como ves, cuando se añade un nivel de indentación (con independencia del número de espacios empleados para ello), se emite un componente **dentro**; cuando se reduce la indentación, se emiten tantos componentes **fuera** como niveles de indentación se retrocede. Al llegar al fin de fichero, se emiten los componentes **fuera** correspondientes a los niveles de indentación no cerrados en ese momento.

Hay un error léxico si, al reducir la indentación, la posición alcanzada no coincide con ninguna de las anteriores. Por ejemplo:

`for i in l:`	**for id in id ptos nl**
`  for j in l:`	**dentro for id in id ptos nl**
`for k in l:`	Error léxico

Puedes asumir que el espacio en blanco de antes de la línea no incluye ningún tabulador y que ningún componente ocupa más de una línea.

Pista: puede ser útil tener un atributo que contenga el número de **fuera** pendientes de emitir y que lo primero que haga el léxico ante una llamada a `siguiente` sea comprobar el valor de ese atributo.

Pregunta 3 (2 puntos)

¿Cuáles de las siguientes afirmaciones acerca de las tablas de análisis LL(1) son ciertas y cuáles falsas? Justifica brevemente las respuestas.

1. No puede haber dos filas no vacías idénticas.
2. No puede haber dos filas vacías.
3. No puede haber dos columnas no vacías idénticas.
4. No puede haber dos columnas vacías.

Duración del examen: 4 horas

¡Buena suerte!