UNIVERSITAT JAUME·I

4º Ingeniería Informática

II26 Procesadores de lenguaje

Examen (10 de septiembre de 2010)

PREGUNTA 1 (5 PUNTOS)

Explica cómo debería modificarse la versión 9 de `minicomp` de modo que aceptara las siguientes extensiones, que son independientes entre sí.

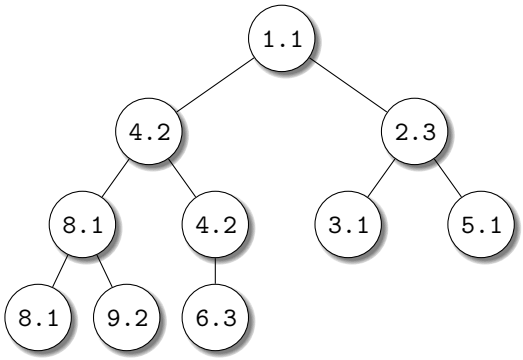
Las descripciones de modificaciones del nivel léxico deben explicitar, si las hay, las nuevas expresiones regulares y los atributos asociados a posibles nuevas categorías, **no utilices categorías sin nombre con metacomp para las categorías que crees**; las de modificaciones del nivel sintáctico o semántico deben aclarar las reglas, acciones semánticas 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. Comprobación de la condición de montículo (2,5 puntos)

Esta extensión permite utilizar la nueva palabra reservada `es_monticulo` como un operador unario, prefijo y de tipo lógico, que tiene como operando un vector *v* de tipo base real y vale cierto si *v* representa un montículo y falso en caso contrario. Por ejemplo, el siguiente fragmento escribiría `si` ya que el árbol representado por el vector (que puedes ver a la derecha) es un montículo:

```
v: vector [10] de real;
...
v[0] = 1.1; v[1] = 4.2; v[2] = 2.3;
v[3] = 8.1; v[4] = 4.2; v[5] = 3.1;
v[6] = 5.1; v[7] = 8.1; v[8] = 9.2;
v[9] = 6.3;
si es_monticulo v entonces
  escribe "si";
si_no
  escribe "no";
fin
```



La prioridad del operador `es_monticulo` coincide con la del operador de cambio de signo.

Recuerda que un vector *v* tiene estructura de montículo si para cada posición *i* mayor o igual que cero se cumple que *v[i]* es menor o igual que sus hijos, que están en *v[2*i+1]* y *v[2*i+2]*, si existen. Alternativamente, puedes considerar que es montículo si para cada *i* mayor que cero, *v[i]* es mayor o igual que su padre, que está en *v[(i-1)/2]*.

Ten en cuenta que el vector puede ser una componente de un vector multidimensional. Por ejemplo:

```
m: vector [10] de vector [20] de real;
...
si es_monticulo m[3] entonces
  escribe "si";
si_no
  escribe "no";
fin
```

Si en el acceso al vector se producen efectos secundarios, estos suceden sólo una vez. Es responsabilidad del programador que el vector haya sido inicializado previamente.

El número de instrucciones generadas debe ser independiente del tamaño del vector.

E2. Operador de pertenencia (2,5 puntos)

Esta extensión permite utilizar la nueva palabra reservada `en` como un operador de tipo lógico para averiguar si un determinado valor está comprendido en un conjunto de valores, todos ellos enteros. El operador tiene la sintaxis

e en [*<lista>*]

donde *e* es una expresión de tipo entero y *<lista>* una lista, separada por comas, de rangos y expresiones enteras. Los rangos están formados por pares de expresiones enteras separadas por el nuevo componente léxico "`..`", que se debe escribir sin ningún espacio o separador entre los dos puntos. El resultado del operador en es cierto si el valor de *e* coincide con el valor de alguna de las expresiones o está incluido en alguno de los rangos de la lista. Un número n está incluido en un rango $a..b$ si $a \leq n \leq b$.

Por ejemplo,

```
x en [6 * 6, 4 + 1 .. 5 + 3, 9 - 9]
```

es cierto si `x` vale `36`, `5`, `6`, `7`, `8` o `0`, y falso en caso contrario.

Por ejemplo, el siguiente código escribe la calidad de una añada de la denominación de origen Penedés:

```
si anyo en [ 1991, 1992, 2002, 2004 ] entonces
  escribe "Buena";
fin
si anyo en [ 1993 .. 1997, 1999 .. 2001 , 2003, 2005 .. 2008 ] entonces
  escribe "Muy buena";
fin
si anyo en [ 1998 ] entonces
  escribe "Excelente";
fin
```

El operador `en` es más prioritario que los operadores relacionales y menos que los aritméticos. Por ejemplo, si `x` vale `12` e `y` vale `6`, la siguiente expresión vale cierto:

```
x+3 en [5+5 .. 40/2] /\ 5*y en [4*y, 30]
```

Los posibles efectos secundarios de evaluar las expresiones deben suceder como máximo una vez.

Pregunta 2 (2 puntos)

Escribe una expresión regular y un AFD para las cadenas largas de Python. Estas se abren con una secuencia de tres comillas simples (`'''`) o de tres comillas dobles (`"""`) y se cierran con la primera aparición de una secuencia de tres caracteres no escapados que coincida con la secuencia de apertura. Entre la apertura y el cierre pueden aparecer cero o más repeticiones de:

- Cualquier carácter ASCII (incluyendo nueva línea y tabulador) salvo una barra invertida aislada.
- Secuencias de escape formadas por el carácter barra invertida seguida de cualquier carácter ASCII (incluyendo nueva línea, tabulador y la propia barra).

Un ejemplo de este tipo de cadenas es:

```
'''Esto es una 'cadena larga' que tiene varias lineas,
que tiene secuencias de escape como \a, \7 y \t
y no se cierra con tres comillas dobles """ sino con
tres simples.'''
```

Importante: En tu solución, debes sustituir `'` por `1` y `"` por `2` para evitar errores de interpretación durante la corrección.

Pregunta 3 (1,5 puntos)

Escribe una gramática con partes derechas regulares que sea RLL(1) y que genere el mismo lenguaje que la siguiente:

$$
\begin{aligned}
\langle \text{Número} \rangle &\rightarrow \langle \text{Octal} \rangle \mid \langle \text{Decimal} \rangle \mid \langle \text{Hexadecimal} \rangle \\
\langle \text{Octal} \rangle &\rightarrow (\langle \text{DígitoOctal} \rangle)^{+}\texttt{o} \\
\langle \text{Decimal} \rangle &\rightarrow (\langle \text{DígitoDecimal} \rangle)^{+}\texttt{d} \\
\langle \text{Hexadecimal} \rangle &\rightarrow (\langle \text{DígitoHexadecimal} \rangle)^{+}\texttt{h} \\
\langle \text{DígitoOctal} \rangle &\rightarrow \texttt{0}\mid\texttt{1}\mid\texttt{2}\mid\texttt{3}\mid\texttt{4}\mid\texttt{5}\mid\texttt{6}\mid\texttt{7} \\
\langle \text{DígitoDecimal} \rangle &\rightarrow \texttt{0}\mid\texttt{1}\mid\texttt{2}\mid\texttt{3}\mid\texttt{4}\mid\texttt{5}\mid\texttt{6}\mid\texttt{7}\mid\texttt{8}\mid\texttt{9} \\
\langle \text{DígitoHexadecimal} \rangle &\rightarrow \texttt{0}\mid\texttt{1}\mid\texttt{2}\mid\texttt{3}\mid\texttt{4}\mid\texttt{5}\mid\texttt{6}\mid\texttt{7}\mid\texttt{8}\mid\texttt{9}\mid\texttt{A}\mid\texttt{B}\mid\texttt{C}\mid\texttt{D}\mid\texttt{E}\mid\texttt{F}
\end{aligned}
$$

Algunos ejemplos de números generados por la gramática son: `123d, 123o, 123h, 789d, 9d, 9h, 28d, 28h, A2h, 1Ah, FFh`.

Pregunta 4 (1,5 puntos)

Considera la siguiente gramática de un sencillo lenguaje ficticio:

$$
\begin{aligned}
\langle \text{Sentencia} \rangle &\rightarrow \textbf{escribe}\ \langle \text{Expresión} \rangle \\
\langle \text{Expresión} \rangle &\rightarrow \langle \text{Término} \rangle \langle \text{RestoExpresión} \rangle \\
\langle \text{RestoExpresión} \rangle &\rightarrow \textbf{arroba}\ \langle \text{Término} \rangle \langle \text{RestoExpresión} \rangle \mid \lambda \\
\langle \text{Término} \rangle &\rightarrow (\langle \text{Prefijo} \rangle)^{*}\ \textbf{número} \\
\langle \text{Prefijo} \rangle &\rightarrow \textbf{almohadilla} \mid \textbf{circunflejo}
\end{aligned}
$$

El terminal **arroba** corresponde a un operador `@` que es binario, infijo y asociativo *por la izquierda*. Los terminales **almohadilla** y **circunflejo** corresponden, respectivamente, a dos operadores `#` y `^`, que son unarios, prefijos y más prioritarios que el operador `@`. Todos ellos tienen operandos y resultado de tipo entero.

Añade las acciones semánticas necesarias para que, al mismo tiempo que se realiza un análisis RLL(1) de la entrada, se calcule y escriba el resultado de evaluar la expresión. Por supuesto, debes tener en cuenta la asociatividad y prioridad de los operadores para obtener el resultado correcto.

Asume que los componentes de la categoría **número** son enteros cuyo valor está en el atributo v, y que dispones también de esos mismos operadores `@`, `#` y `^` en el lenguaje de las acciones semánticas.

Ten en cuenta que no puedes modificar la gramática, que no puedes utilizar variables globales, que todos los atributos que utilices han de ser de tipo entero o lógico (este requisito no afecta a las posibles variables locales) y que ningún atributo puede ser heredado en un sitio y sintetizado en otro. Indica para cada atributo si es heredado o sintetizado y qué representa.

© Universitat Jaume I 2009-2010