Ingeniería Informática
Procesadores de lenguaje
Examen de teoría (28 de mayo de 2005 — Solución)

Introducción

Lo que sigue es una posible solución al examen del día 28 de mayo de 2005. Obviamente, son posibles soluciones alternativas. Además, hay bastante explicación que no se espera en vuestras respuestas (esto es especialmente notable en el caso de las expresiones regulares).

PREGUNTA 1 (5 PUNTOS)

Operadores mínimo y máximo (2 puntos)

En el analizador léxico, podemos añadir una o dos categorías léxicas. Si optamos por una, por ejemplo **mm**, tendríamos:

categoría	expresión regular	acciones	atributos
mm	[<>]\?	copiar lexema emitir	lexema

Con dos categorías, nos "ahorramos" el lexema:

categoría	expresión regular	acciones	atributos
mín	<\?	emitir	—
máx	>\?	emitir	—

En la parte sintáctica, tendremos que añadir una regla entre las de igualdad e y-lógico. Supondremos una estructura similar a:

$$\begin{aligned} \langle \text{Ylogico} \rangle &\rightarrow \langle \text{Igualdad} \rangle (\&\& \langle \text{Igualdad} \rangle) * \\ \langle \text{Igualdad} \rangle &\rightarrow \langle \text{Orden} \rangle (\textbf{opigu} \langle \text{Orden} \rangle) * \end{aligned}$$

Lo cambiamos a:

$$\begin{aligned} \langle \text{Ylogico} \rangle &\rightarrow \langle \text{MaxMin} \rangle (\&\& \langle \text{MaxMin} \rangle) * \\ \langle \text{MaxMin} \rangle &\rightarrow \langle \text{Igualdad} \rangle (\textbf{mm}\ \langle \text{Igualdad} \rangle) * \\ \langle \text{Igualdad} \rangle &\rightarrow \langle \text{Orden} \rangle (\textbf{opigu}\ \langle \text{Orden} \rangle) * \end{aligned}$$

Si tuviéramos dos categorías, sería:

$$\begin{aligned} \langle \text{Ylogico} \rangle &\rightarrow \langle \text{MaxMin} \rangle (\&\& \langle \text{MaxMin} \rangle) * \\ \langle \text{MaxMin} \rangle &\rightarrow \langle \text{Igualdad} \rangle ((\textbf{max}|\textbf{min}) \langle \text{Igualdad} \rangle) * \\ \langle \text{Igualdad} \rangle &\rightarrow \langle \text{Orden} \rangle (\textbf{opigu}\ \langle \text{Orden} \rangle) * \end{aligned}$$

Para la parte semántica, utilizaremos el nodo NodoMaxMin, que tendrá dos hijos, i y d, representando los operandos izquierdo y derecho, y un atributo, op, que guardará la operación. Las acciones semánticas son:

$$\begin{aligned}\langle \text{MaxMin}\rangle \quad \rightarrow \quad & \langle \text{Igualdad}\rangle_1 \ \{\text{arb:= Igualdad}_1\text{.arb;}\} \\ & (\textbf{mm}\ \langle \text{Igualdad}\rangle_2 \ \{\text{arb:= NodoMaxMin(arb, nm.lexema, } \langle \text{Igualdad}\rangle_2\text{.arb);}\}) * \\ & \{\langle \text{MaxMin}\rangle\text{.arb:= arb;}\}\end{aligned}$$

Con dos categorías:

$$\begin{aligned}\langle \text{MaxMin}\rangle \quad \rightarrow \quad & \langle \text{Igualdad}\rangle_1 \ \{\text{arb:= Igualdad}_1\text{.arb; }\} \\ & ((\textbf{max}\ \{\text{op:= ``>?''}\}|\textbf{min}\ \{\text{op:= ``<?''}\}) \\ & \quad \langle \text{Igualdad}\rangle_2 \ \{\text{arb:= NodoMaxMin(arb, op, } \langle \text{Igualdad}\rangle_2\text{.arb; }\}) * \\ & \{\langle \text{MaxMin}\rangle\text{.arb:= arb}\}\end{aligned}$$

Las comprobaciones semánticas en NodoMaxMin comenzarían por llamar a las comprobaciones de los hijos izquierdo y derecho. Se produciría un error si alguno de ellos no tuviera tipo aritmético. Si ambos son de tipo aritmético, se elegiría como tipo del nodo el más general y, si fuera necesario, se añadiría un nodo de conversión al nodo menos general.

Para generar código, tenemos que fijarnos en el tipo del nodo. Tendremos dos posibilidades: que sea de tipo entero o de tipo real.

Si el tipo es entero, podemos generar código Stan de la siguiente forma[1]:

```
fin= etiquetas.nueva()
self.i.genera_codigo(c)
self.d.genera_codigo(c)
c.append(POP(MD(memoria.TMP), "TMP")) # Derecho en TMP
c.append(DUP())
c.append(PUSH(MD(memoria.TMP), "TMP")) # Pila: izdo, izdo, dcho
if self.op== "<?":
  c.append(Comparacion("<"))
else:
  c.append(Comparacion(">"))
# Después de la comparación, en la pila tenemos el izdo
c.append(BRD(fin)) # si el elegido es el izdo, hemos terminado
c.append(POP())    # si no, ponemos el derecho en el tope de la pila
c.append(PUSH(MD(memoria.TMP), "TMP"))
c.append(Etiqueta(fin))
```

El código Rossi sería:

```
fin= etiquetas.nueva()
self.i.genera_codigo(c)
iz= registros.actual()
self.d.generaCodigo(c)
de= registros.actual()
registros.libera() # El resultado lo dejaremos en el registro iz
# Saltamos a fin si ya está
if self.op== "<?":
  c.append(R.ble(iz, de, fin))
else:
  c.append(R.bge(iz, de, fin))
# Si no, guardamos de en iz
c.append(R.add(iz, de, "zero"))
c.append(R.Etiqueta(fin))
```

[1]Seguimos el esquema de minicomp.

El caso de los reales es similar. Para Stan:

```
fin= etiquetas.nueva()
self.i.genera_codigo(c)
self.d.genera_codigo(c)
c.append(FPOP(MD(memoria.TMP), "TMP")) # Derecho en TMP
c.append(FDUP())
c.append(FPUSH(MD(memoria.TMP), "TMP")) # Pila: izdo, izdo, dcho
if self.op== "<?":
  c.append(Comparacion("F<"))
else:
  c.append(Comparacion("F>"))
# Después de la comparación, en la pila tenemos el izdo
c.append(BRD(fin)) # si el elegido es el izdo, hemos terminado
c.append(FPOP())   # si no, ponemos el derecho en el tope de la pila
c.append(FPUSH(MD(memoria.TMP), "TMP"))
c.append(Etiqueta(fin))
```

Y en Rossi[2]:

```
fin= etiquetas.nueva()
self.i.genera_codigo(c)
iz= registros.actualr()
self.d.generaCodigo(c)
de= registros.actualr()
registros.liberar() # El resultado lo dejaremos en el registro iz
# Saltamos a fin si ya está
if self.op== "<?":
  c.append(R.fble(iz, de, fin))
else:
  c.append(R.fbge(iz, de, fin))
c.append(R.fadd(iz, de, "fzero"))
c.append(R.Etiqueta(fin))
```

El código que se generaría para

```
f= i<?c;
```

sería:

- En Stan:

```
    PUSH &10 # i
    PUSH &11 # c
    POP  &2  # TMP
    DUP
    PUSH &2  # TMP
    <
    BRD et1
    POP
    PUSH &2  # TMP
  et1:
    TOFLOAT
    FPOP &12 # f
```

- En Rossi:

```
    lw $r0, 10($zero) # i
    lw $r1, 11($zero) # c
    ble $r0, $r1, et1
    add $r0, $r1, $zero
  et1:
    tofloat $fr0, $r0
    fsw $fr0, 12($zero) # f
```

[2]Asumimos que `actualr` y `liberar` son análogas a `actual` y `libera` para controlar los registros reales.

Máximos de secuencias (3 puntos)

En el nivel léxico, nos basta con añadir, antes de la categoría identificador, la categoría **max** con expresión regular `max`, sin atributos y con emitir como acción.

En el nivel sintáctico, añadiríamos, en la regla de máxima prioridad, el nuevo operador:

⟨Elemental⟩ → **max abre** ⟨Expresión⟩ **pyc** ⟨Expresión⟩ **pyc** ⟨Expresión⟩ **cierra**

En el nivel semántico, utilizaremos el NodoMáximo. Tendrá tres hijos, e1, e2 y e3, para cada una de las expresiones. Para las comprobaciones semánticas, bastará comprobar que todas las expresiones son de tipos elementales y como tipo del nodo tomaremos el de e3.

Nuevamente, la generación de código dependerá del tipo.

Para el tipo entero, podemos generar código Stan así:

```
fin= etiquetas.nueva()
cond= etiquetas.nueva()
sigue= etiquetas.nueva()
self.e1.generaCodigo()
if self.e1.tipo== tipos.Real:
  c.append(FPOP())
else:
  c.append(POP())
self.e2.codigoControl(None, fin) # El comportamiento si no entramos
                                 # está indefinido
self.e3.generaCodigo()
c.append(JMP(cond))
c.append(Etiqueta(sigue))
c.append(DUP())
self.e3.generaCodigo()
c.appned(DUP())
c.append(POP(MD(memoria.TMP)), "TMP") # Pila: max, max, e3
c.append(Comparacion(">="))
c.append(BRD(cond))
c.append(POP()) # Tenemos un nuevo máximo
c.append(PUSH(MD(memoria.TMP), "TMP"))
c.append(Etiqueta(cond))
self.e2.codigoControl(sigue, None)
c.append(Etiqueta(fin))
```

En el caso del tipo real:

```
fin= etiquetas.nueva()
cond= etiquetas.nueva()
sigue= etiquetas.nueva()
self.e1.generaCodigo()
if self.e1.tipo== tipos.Real:
  c.append(FPOP())
else:
  c.append(POP())
self.e2.codigoControl(None, fin) # El comportamiento si no entramos
                                 # está indefinido
self.e3.generaCodigo()
c.append(JMP(cond))
c.append(Etiqueta(sigue))
c.append(FDUP())
self.e3.generaCodigo()
c.append(FDUP())
c.append(FPOP(MD(memoria.TMP), "TMP")) # Pila: max, max, e3
c.append(Comparacion("F>="))
```

```
c.append(BRD(cond))
c.append(FPOP()) # Tenemos un nuevo máximo
c.append(FPUSH(MD(memoria.TMP), "TMP"))
c.append(Etiqueta(cond))
self.e2.codigoControl(sigue, None)
c.append(Etiqueta(fin))
```

El código Rossi para el tipo entero sería:

```
fin= etiquetas.nueva()
cond= etiquetas.nueva()
sigue= etiquetas.nueva()
self.e1.generaCodigo()
if self.e1.tipo== tipos.Real:
  registros.liberar()
else:
  registros.libera()
self.e2.codigoControl(None, fin)
self.e3.generaCodigo()
m= registros.actual() # el registro m tiene el máximo
c.append(R.j(cond))
c.append(R.Etiqueta(sigue))
self.e3.generaCodigo()
aux= registros.actual()
registros.libera()
c.append(R.bge(m, aux, cond))
c.append(R.add(m, aux, "zero")) # Tenemos un nuevo máximo
c.append(R.Etiqueta(cond))
self.e2.codigoControl(sigue, None)
c.append(R.Etiqueta(fin))
```

En el caso del tipo real:

```
fin= etiquetas.nueva()
cond= etiquetas.nueva()
sigue= etiquetas.nueva()
self.e1.generaCodigo()
if self.e1.tipo== tipos.Real:
  registros.liberar()
else:
  registros.libera()
self.e2.codigoControl(None, fin)
self.e3.generaCodigo()
m= registros.actualr() # el registro m tiene el máximo
c.append(R.j(cond))
c.append(R.Etiqueta(sigue))
self.e3.generaCodigo()
aux= registros.actualr()
registros.liberar()
c.append(R.fbge(m, aux, cond))
c.append(R.fadd(m, aux, "fzero")) # Tenemos un nuevo máximo
c.append(R.Etiqueta(cond))
self.e2.codigoControl(sigue, None)
c.append(R.Etiqueta(fin))
```

El código que se generaría para[3]

```
printf("El doble del máximo es: %f", 2*max(i=0; i<n; v[i++]));
```

sería (con algunos comentarios adicionales para entenderlo):

- En Stan

[3]Tal como estaba el ejemplo original, se entraba en un bucle infinito, así que se cambió al que vemos aquí.

```
# Escritura de la cadena
PUSH 10 # "El doble del máximo es: "
CALL escribe
# Producto
PUSH 2  # Primer operando
TOFLOAT # Promoción
# Comienza el max
#   Asignación i=0
PUSH 0
DUP
POP &100 # i
POP
#   Comparación i<n
PUSH &100 # i
PUSH &101 # n
<
BRZ et1
#   Acceso v[i++]
PUSH 102 # v
#     Cálculo i++
PUSH 100 # i
POP &2 # TMP
PUSH 0[&2]
INC 0[&2]
ADD
POP &2 # TMP
FPUSH 0[&2]
#  Salto a la condición
JMP et2
et3:
  FDUP # Duplicamos el máximo actual
  #   Acceso v[i++]
  PUSH 102 # v
  #     Cálculo i++
  PUSH 100 # i
  POP &2 # TMP
  PUSH 0[&2]
  INC 0[&2]
  ADD
  POP &2 # TMP
  FPUSH 0[&2]
  #  Cambiamos el máximo si hace falta
  FDUP
  FPOP &2 # TMP
  F>=
  BRD et2
  FPOP
  FPUSH &2 # TMP
et2:
  #  Comparación i<n
  PUSH &100 # i
  PUSH &101 # n
  <
  BRD et3
et1:
  FMUL # Multiplicación
  FOUT # Escritura
```

- En Rossi

```
# Escritura de la cadena
addi $r0, $zero, 10 # "El doble..."
add $a0, $r0, $zero
addi $sc, $zero, 2  # escribe cadena
syscall
# Producto
addi $r0, $zero, 2 # Primer operando
tofloat $f0, $r0   # Promoción
# Comienza el max
#  Asignación i=0
addi $r0, $zero, 0
sw $r0, 100($zero) # i
#  Comparación i<n
lw $r0, 100($zero) # i
lw $r1, 101($zero) # n
bge $r0, $r1, et1
#  Acceso v[i++]
addi $r0, $zero, 102 # v
#   Cálculo i++
addi $r3, $zero, 100 # i
lw $r1, 0($r3)
lw $r2, 0($r3)
addi $r2, $r2, 1
sw $r2, 0($r3)
add $r0, $r0, $r1
flw $f1, 0($r0)
  #  Salto a la condición
  j et2
et3:
  #  Acceso v[i++]
  addi $r0, $zero, 102 # v
  #   Cálculo i++
  addi $r3, $zero, 100 # i
  lw $r1, 0($r3)
  lw $r2, 0($r3)
  addi $r2, $r2, 1
  sw $r2, 0($r3)
  add $r0, $r0, $r1
  flw $f2, 0($r0)
  #  Cambiamos el máximo si hace falta
  fbge $f1, $f2, et2
  fadd $f1, $f2, $zero
et2:
  #  Comparación i<n
  lw $r0, 100($zero) # i
  lw $r1, 101($zero) # n
  ble $r0, $r1, et3
et1:
  fmul $f0, $f0, $f1 # multiplicación
  fadd $fa, $f0, $fzero
  addi $sc, $zero, 1 # escribe real
  syscall
```

PREGUNTA 2 (2 PUNTOS)

Cadenas de dígitos que no empiecen por 22. Para que la cadena no empiece por 22, tenemos dos opciones: o bien comienza por algo que no sea un dos[4]: $\langle[\hat{}2].^*\rangle$, o bien empieza por un dos seguido de algo que no es un dos y lo que sea: $\langle 2[\hat{}2].^*\rangle$. Además, puede suceder que la cadena sea vacía[5] o justamente 2. En total, tenemos $\langle\lambda|2|[\hat{}2].^*|2[\hat{}2].^*\rangle$. Podemos manipular la expresión para llegar a $\langle 2?([\hat{}2].^*)?\rangle$.

Cadenas de dígitos que no empiecen ni terminen por 22. En este caso, tenemos que hacer algo similar a lo de antes, pero sustituyendo las "colas" $\langle.^*\rangle$ por algo que impida el 22. Por un lado, la cadena puede terminar en algo distinto de dos; por otro, puede terminar en dos si el anterior no lo era. Con esto, el núcleo de la expresión queda $\langle([\hat{}2]|2[\hat{}2]).^*([\hat{}2]|[\hat{}2]2)\rangle$. A ese núcleo le faltan algunas cadenas de longitud cero, uno, dos y tres. Completamos con: $\langle\lambda|.|[\hat{}2].|.[\hat{}2]|.[\hat{}2].\rangle$. Así, la expresión queda: $\langle\lambda|.|[\hat{}2].|.[\hat{}2]|.[\hat{}2].|([\hat{}2]|2[\hat{}2]).^*([\hat{}2]|[\hat{}2]2)\rangle$ y la podemos retocar ligeramente: $\langle.?([\hat{}2].?)?|2?[\hat{}2].^*[\hat{}2]2?\rangle$.

Cadenas de dígitos que no contengan la subcadena 22. Podemos emplear la expresión de los apuntes para las comentarios que terminan en >> retocándola ligeramente. La expresión tal cual sería $\langle(2?[\hat{}2])^*\rangle$. El retoque consiste en permitir que la cadena termine en dos: $\langle(2?[\hat{}2])^*2?\rangle$.

Cadenas de dígitos que contengan exactamente una vez la subcadena 22. Podemos intentar construir esta expresión a partir de la anterior concatenando dos copias con la cadena 22: $\langle(2?[\hat{}2])^*22(2?[\hat{}2])^*\rangle$. Sin embargo, con esto tenemos el problema de permitir que 222, que tiene dos copias de 22, aparezca en la cadena. Lo que podemos hacer es coger la parte que no tiene ningún 22 y concatenarla antes y después, teniendo en cuenta que después no puede empezar por dos pero sí terminar por él (podemos "darle la vuelta"): $\langle(2?[\hat{}2])^*22([\hat{}2]2?)^*\rangle$.

PREGUNTA 3 (1,5 PUNTOS)

Utilizaremos, para ⟨N⟩ y ⟨A⟩ un atributo adicional, p, que nos dirá si la cadena generada tiene longitud par o impar. En el caso de ⟨Dig⟩, utilizamos el atributo v con el valor del dígito (que supondremos que está en el atributo valor de los componentes de la categoría **dígito**).

Las acciones de las reglas de ⟨N⟩ son:

⟨N⟩ → ⟨Dig⟩$_1$⟨Dig⟩$_2$⟨A⟩$_1$⟨Dig⟩$_3$⟨A⟩$_2$
{si:= ⟨Dig⟩$_1$.v+⟨A⟩$_1$.si;
sp:= ⟨Dig⟩$_2$.v+⟨A⟩$_1$.sp;
si ⟨A⟩$_1$.p **entonces**
si:= si+ ⟨Dig⟩$_3$.v+⟨A⟩$_2$.sp; sp:= sp+⟨A⟩$_2$.si;
si no
si:= si+ ⟨A⟩$_2$.si; sp:= sp+ ⟨Dig⟩$_3$.v+⟨A⟩$_2$.sp;
fin si
⟨N⟩.si:= si; ⟨N⟩.sp:= sp; ⟨N⟩.p:= ⟨A⟩$_1$.p≠⟨A⟩$_2$.p;}

⟨N⟩ → ⟨N⟩$_1$⟨Dig⟩⟨A⟩
{**si** ⟨N⟩$_1$.p **entonces**
si:= ⟨N⟩$_1$.si+⟨Dig⟩.v+⟨A⟩.sp;
sp:= ⟨N⟩$_1$.sp+⟨A⟩.si;
si no
si:= ⟨N⟩$_1$.si+⟨A⟩.si;
sp:= ⟨N⟩$_1$.sp+⟨Dig⟩.v+ ⟨A⟩.sp;
fin si
⟨N⟩.si:= si; ⟨N⟩.sp:= sp; ⟨N⟩.p:= ⟨N⟩$_1$.p≠⟨A⟩.p;}

[4]Por claridad, hemos encerrado entre paréntesis angulares las expresiones regulares.

[5]Como puede ser más o menos dudoso que la cadena vacía sea aceptable, al corregir tomaremos las dos posibilidades (se acepta la cadena vacía o no) como válidas.

Las de ⟨A⟩:

$$
\begin{array}{lcl}
\langle \mathsf{A} \rangle & \rightarrow & \langle \mathsf{A} \rangle_1 \langle \mathsf{A} \rangle_2 \\
 & & \{\textbf{si } \langle \mathsf{A} \rangle_1.\mathrm{p} \textbf{ entonces} \\
 & & \quad \langle \mathsf{A} \rangle.\mathrm{si} := \langle \mathsf{A} \rangle_1.\mathrm{si} + \langle \mathsf{A} \rangle_2.\mathrm{si}; \\
 & & \quad \langle \mathsf{A} \rangle.\mathrm{sp} := \langle \mathsf{A} \rangle_1.\mathrm{sp} + \langle \mathsf{A} \rangle_2.\mathrm{sp}; \\
 & & \textbf{si no} \\
 & & \quad \langle \mathsf{A} \rangle.\mathrm{si} := \langle \mathsf{A} \rangle_1.\mathrm{si} + \langle \mathsf{A} \rangle_2.\mathrm{sp}; \\
 & & \quad \langle \mathsf{A} \rangle.\mathrm{sp} := \langle \mathsf{A} \rangle_1.\mathrm{sp} + \langle \mathsf{A} \rangle_2.\mathrm{si}; \\
 & & \textbf{fin si} \\
 & & \langle \mathsf{A} \rangle.\mathrm{p} := \langle \mathsf{A} \rangle_1.\mathrm{p} = \langle \mathsf{A} \rangle_2.\mathrm{p};\} \\
\langle \mathsf{A} \rangle & \rightarrow & \langle \mathsf{Dig} \rangle \ \{\langle \mathsf{A} \rangle.\mathrm{si} := \langle \mathsf{Dig} \rangle.\mathrm{v};\ \langle \mathsf{A} \rangle.\mathrm{sp} := 0;\ \langle \mathsf{A} \rangle.\mathrm{p} := \mathrm{falso};\}
\end{array}
$$

Y las de ⟨Dig⟩

$$\langle \mathsf{Dig} \rangle \rightarrow \textbf{digito } \{\langle \mathsf{Dig} \rangle.\mathrm{v} := \textbf{digito}.\mathrm{valor}\}$$

Pregunta 4 (1,5 puntos)

Si G es LL(1), entonces G^{-1} es LL(1). Es fácil ver que la gramática G con reglas

$$\langle \mathsf{S} \rangle \rightarrow \mathsf{a\ b} | \mathsf{b\ b}$$

es LL(1) y que G^{-1} no lo es. Así pues, la afirmación es falsa.

Si G es LL(1), entonces G^{-1} no es LL(1). Nuevamente es fácil encontrar un contraejemplo. La gramática que tiene como única regla $\langle \mathsf{S} \rangle \rightarrow \mathsf{a}$ es LL(1), al igual que su inversa (que coincide con ella). Así pues, la afirmación es falsa.

Si G es SLR, entonces G^{-1} es SLR. Consideremos la gramática G con reglas

$$
\begin{array}{lcl}
\langle \mathsf{S} \rangle & \rightarrow & \mathsf{a}\langle \mathsf{A} \rangle | \mathsf{b\ a}\langle \mathsf{B} \rangle \\
\langle \mathsf{A} \rangle & \rightarrow & \mathsf{a} \\
\langle \mathsf{B} \rangle & \rightarrow & \mathsf{a}
\end{array}
$$

Dibujando los correspondientes autómatas, se ve que G es SLR y que G^{-1} no es SLR. La afirmación es falsa.

Si G es SLR, entonces G^{-1} no es SLR. Podemos utilizar el mismo contraejemplo que en el segundo caso. También en este caso, la afirmación es falsa.