Universidad de Costa Rica

Escuela de Ciencias de la Computación e Informática

CI-2700

TÓPICOS ESPECIALES - COMPILADORES

II Ciclo 2014

Profesor: Manuel E. Bermúdez

EXAMEN FINAL

(NO PARA COMER EN CLASE, SINO PARA LLEVAR)

30 horas, comenzando 04/12/14 12m, terminando 05/12/14 6 p.m.

Problema 1_____(20p.)

Problema 2_____(20p.)

Problema 3_____(20p.)

Problema 4_____(20p.)

Problema 5_____(20p.)

TOTAL _____(100p.)

NOMBRE_____

FAVOR ENTREGAR ESTE EXAMEN EN FORMA ELECTRÓNICA

PROBLEMA 1. (20 pts.)

La siguiente gramática de transducción describe un lenguaje de expresiones regulares sobre el alfabeto {a,b,c}

```
E0 -> E0 '|' E1     => '|'
   -> E1
E1 -> E2 E1         => 'cat'
   -> E2
E2 -> E2 'list' E3  => 'list'
   -> E3
E3 -> E4 '+'        => '+'
   -> E4 '*'        => '*'
   -> E4 '?'        => '?'
   -> E4
E4 -> '(' E0 ')'
   -> 'a'           => 'a'
   -> 'b'           => 'b'
   -> 'c'           => 'c'
```

Esta es una expresión regular de muestra:

```
(a* | b)* list c a+
```

a) Mostrar que la gramática no es LL(1).

b) Transformar la gramática para que sea LL(1), calcular conjuntos Select, y mostrar que la nueva gramática es LL(1).

c) Mostrar la tabla de análisis sintáctico de la gramática modificada.

d) Mostrar, paso a paso, el análisis sintáctico de la expresión regular mostrada arriba, utilizando la tabla del paso (c). En cada paso, mostrar la pila, la entrada, y cuando sea apropiado, el valor obtenido de la tabla.

e) Escribir un analizador sintáctico de descenso recursivo para este lenguaje, agregando instrucciones 'BuildTree' para construir el árbol de sintaxis abstracta en forma ascendente, según la gramática original. Sugerencia: Powerpoint "Análisis Sintáctico", transparencias 116-119.

PROBLEMA 2. (20 pts.)

Considere la misma gramática que en el Problema 1.

a) Construir los conjuntos de items LR(0), y construir las tablas ACCION y GOTO del analizador sintáctico LR(0).

b) Mostrar que la gramática no es LR(0).

c) Determinar si la gramática es SLR(1). Mostrar el análisis necesario.

d) Determinar si la gramática es LALR(1). Mostrar el análisis necesario.

e) Usando sus tablas ACCION y GOTO, mostrar, paso a paso, el análisis sintáctico de la expresión regular mostrada en el Problema 1. En cada paso, mostrar la pila, la entrada, y la acción tomada (desplazamiento o reducción). Luego, tomar la secuencia de reducciones realizada por la analizador, y utilizarla para construir, en forma ascendente, el árbol de Sintaxis Abstracta para la hilera de entrada.

PROBLEMA 3. (20 pts.)

Considere la siguiente gramática libre de contexto:

```
S ->E '|'
E ->A a
  ->b A c
  ->d c
  ->b d a
A ->d
```

a) Construir los conjuntos de items LR(0).

b) Mostrar que la gramática no es LR(0). Mostrar el análisis necesario.

c) Mostrar que la gramática no es SLR(1). Mostrar el análisis necesario.

d) Mostrar que la gramática es LALR(1). Mostrar el análisis necesario.

PROBLEMA 4. (20 pts.)

En esta pregunta, y la siguiente, vamos a construir un compilador-interpretador para números romanos. Abajo aparece una gramática, ligeramente incompleta, que reconoce una lista de números romanos, separados por commas. Cada número varía de 1 a 3999. Por si no recuerda cómo funcionan los números romanos, las siguientes reglas debieran refrescarle la memoria.

1) Los números romanos se componen de los símbolos I (uno), V (cinco), X (diez), L (cincuenta), C (cien), D (quinientos), y M (mil).
2) Un símbolo menor que precede a uno mayor es restado de él. Por ejemplo, IV significa cuatro. Solo una resta de este tipo se permite a la vez, (e.g. IIV no significa tres). Se aplican otras restricciones (ver la gramática).
3) Un símbolo menor (o igual) que sigue a uno mayor (o igual), se le suma. Así, DC significa 600, XIII significa 13, y MMM significa 3000). De nuevo, se aplican otras restricciones.

Romans	→	Roman (’,’ Roman)*	=> ’romans’
Roman	→	Thous Hunds Tens Ones	=> ’ ’
Thous	→	M? M? M?	=> ’ ’
Hunds	→	C C? C?	=> ’ ’
	→	C (D | M)	=> ’ ’
	→	D C? C? C?	=> ’ ’
	→		
Tens	→	X X? X?	=> ’ ’
	→	X (L | C)	=> ’ ’
	→	L X? X? X?	=> ’ ’
	→		
Ones	→	I I? I?	=> ’ ’
	→	I (V | X)	=> ’ ’
	→	V I? I? I?	=> ’ ’
	→		
M	→	’M’	=> ’1000’
D	→	’D’	=> ’500’
C	→	’C’	=> ’100’
L	→	’L’	=> ’50’
X	→	’X’	=> ’10’
V	→	’V’	=> ’5’
I	→	’I’	=> ’1’

a) Algunas de las partes de traducción están en blanco. Cada una debe ser "+" o "-". FAVOR LLENARLAS.

b) Mostrar el árbol de derivación, y el árbol de sintaxis abstracta, para la siguiente hilera de entrada:

DLVII, MI, MCMXCIV, MMXIV

c) Abajo aparece un programa, en pseudo-código, que completa la traducción de la lista de números romanos. Hace falta el código de los casos "Plus" y "Minus". FAVOR LLENAR-LOS.

```
program RomanBackEnd(input,output);
const
     One = '1';
     Five = '5';
     Ten  = '10';
     Fifty = '50';
     Hundr = '100';   { Nombres de Nodos }
     FiveH = '500';
     Thous = '1000';
     Plus  = '+';
     Minus = '-';
var i : integer;
function Traverse (T:TreeNode): integer;
var N,i: integer;
begin
     case NodeName(T) of
          Plus:
          Minus:
          One   : return(1);
          Five  : return(5);
          Ten   : return(10);
          Fifty : return(50);
          Hundr : return(100);
          FiveH : return(500);
          Thous : return(1000);
     end
end;
begin {main}
     ReadTree;
     for i := 1 to Rank(RootOfTree(1)) do
          writeln('Value Is ',Traverse (Child(Root,i)));
end.
```

d) En su árbol de sintaxis abstracta, anotar cada nodo "+" y "-" con el valor retornado por la función "Traverse", al ser aplicada a ese nodo. Mostrar la salida del programa, para la hilera de entrada mostrada arriba.

PROBLEMA 5. (20 pts.)

Implementar el lenguaje de números romanos, utilizando el TWS.

1) Comenzar con una copia nueva (recién descargada y compilada) del TWS.

2) Renombrar ’tiny’ como ’romans’. En el directorio ’romans’, eliminar todo lo relacionado con el generador de código, y el directorio ’tests’. Eliminar los archivos ’tc...’, excepto ’tc’. Renombrar ’tc’ como ’rc’. Editar "rc" de modo que haga el análisis sintáctico, y que luego corra ’Romans’, en lugar de ’Constrain’. No se olvide de cambiar el mensaje de "YAHOO!" :-)

3) En el directorio ’parser’, renombrar ’lex.tiny’ y ’parse.tiny’ como ’lex.romans’ y ’parse.romans’, respectivamente. Modificar estos archivos para reconocer números romanos. Modificar ’Makefile’ en forma corresondiente. Nota: el TWS parece tener problemas con expresiones regulares como M M? M?. Descomponerlos así:

```
Thous -> M
      -> M M
      -> M M M
```

4) En el directorio ’romans’, renombrar ’Constrainer.c’ como ’Romans.c’. Eliminar ’Constrain’. Editar el archivo ’Makefile’ de forma que compile ’Romans.c’ (generando el ejecutable ’Romans’) en lugar de compilar ’Constrainer.c’. Eliminar todas las referencias al generador de código.

5) Ahora, editar ’Romans.c’. Eliminar ’AddIntrinsics’ y cualquier llamdo a esa función. Cambiar la lista de nombres de nodos. En la rutina ’ProcessNode’, manejar el nodo ’romans’, llamando ’Expression’ sobre cada hijo, e imprimiendo los valores retornados por ’Expression’. Expression cumple la función de ’Traverse’ en la pregunta 4). En la función ’Expression’, manejar todos los demás nodos, retornando un valor entero en cada caso.

6) Favor entregar, por correo electrónico, en un comprimido .zip, los archivos ’lex.romans’, ’parse.romans’, y ’Romans.c’, junto con el resto de su examen resuelto.