Ejercicios DLP

Conceptos Básicos del Diseño de Lenguajes – Tema 1

Actividades y preguntas

1) ¿Cuál es la diferencia entre compiladores e intérpretes?
2) ¿Cuáles son las principales responsabilidades de las siguientes fases en un procesador de lenguaje?
    a. Análisis Léxico
    b. Análisis sintáctico
    c. Análisis semántico
    d. Generación de Código
    e. Optimización de código
3) ¿Cuál es el patrón estructural más común usado en la implementación de un lenguaje de programación?
4) ¿Cuál es la principal diferencia entre un árbol de análisis y un árbol de sintaxis abstracta (AST)?
5) ¿Cuál es la diferencia entre un compilador y un transpilador?
6) Describe 4 ejemplos distintos de procesadores de lenguaje diferentes al compilador, intérprete y transpilador
7) Identifica 3 usos diferentes del diseño de lenguajes de programación, distintos al diseño y la implementación de los procesadores de lenguaje
8) Define front, middle y back-en de un compilador
9) ¿Cuál es la principal diferencia entre un compilador y un traductor de lenguaje?
10) ¿Qué fase del procesador del lenguaje realiza conversiones implícitas de expresiones?
11) ¿Qué fase del procesador del lenguaje comprueba que el operador – puede aplicarse a una única expresión?
12) Identifica las fases con las siguientes responsabilidades:
    a. Hacer que el código final se ejecute rápidamente
    b. Transformar bucles en saltos
    c. Añadir información al AST
    d. Comprobar la aridad (p.ej, el número de parámetros) de un operador
    e. Representar el código de entrada en memoria
    f. Detectar el número correcto de argumentos en la invocación a una función
    g. Avisar al programador del uso de caracteres que no están contemplados en el lenguaje
    h. Asignar offsets (desplazamientos) a variables en el programa de entrada
    i. Aprovechar al máximo el uso de los registros de la CPU
    j. Comprobar si se puede realizar una conversión implícita
13) Nombra la información pasada entre:
    a. El lexer y el parser
    b. El parser y el analizador semántico
    c. El analizador semántico y la generación de código

Análisis Léxico – Tema 2

Actividades y preguntas

1) Escribe una Context-Free-Grammar (CFG) para reconocer el siguiente lenguaje:
   {n (A| B)m] n : n,m≥0}, where VT = { A, B, [,] }
2) ¿Qué reconoce el siguiente patrón léxico de ANTLR? `3.7`
3) Escribe un archivo de especificación léxica ANTLR para reconocer los siguientes patrones:
   a. Identificadores: `var1`, `a`, `var_2`, `__private`, `_`
   b. Constantes Reales con punto flotante: `12.3`, `2.`, `.34`
   c. Constantes Reales con mantisa y exponente: `34.12E-3`, `3e3`, `.3E+3`, `2.e23`
   d. Descartar comentarios de línea: `// This is one single-line comment`
4) En ANTLR, ¿qué sucede si un lexema coincide con dos patrones léxicos?
5) Define derivación en un paso
6) Define cadena de idioma
7) ¿Cuál es la principal diferencia entre las CFGs y las expresiones regulares?
8) Define gramática libre de contexto (CFG)
9) En ANTLR, ¿qué sucede si un lexema no es reconocido por ningún patrón?
10) ¿Qué reconoce el siguiente patrón léxico de ANTLR? `[3-7d-f]`
11) Define expresión regular
12) ¿Qué reconoce el siguiente patrón léxico de ANTLR? `[A|B|C]`
13) Escribe una CFG para una posible secuencia vacía de `A` terminales separados por el terminal coma (,)
14) Escribe una producción léxica ANTLR para constantes enteras

Análisis sintáctico - Tema 3

Actividades y preguntas

1) ¿Cuál es la principal relación entre un parse tree y un AST?
2) Define gramática ambigua. ¿Por qué debe evitarse?
3) Define:
   a. Derivación en un paso
   b. Derivación
   c. Cadena
   d. Programa o sentencia
4) ¿Qué recursión debe evitarse en los LL parsers? ¿Por qué?
5) Di si las siguientes gramáticas son ambiguas o no. ¿Por qué?
   a. statement → if-stmt
      statement → ID = exp
      expression → ID
      expression → INT_CONSTANT
      if-stmt → IF (exp) statement
      if-stmt → IF (exp) statement ELSE statement

b. expression → expression + expression
   expression → ID
   expression → INT_CONSTANT
c. expression → (expression)
   expression → - expression
   expression → ID
   expression → INT_CONSTANT

6) ¿Qué hace ANTLR si la entrada de una gramática es ambigua?
7) En ANTLR, ¿qué es devuelto por un método que implementa una producción cuando no hay código embebido añadido?
8) Dada la siguiente gramática:

   e → A e B
   e → ε

   escriba el orden de creación de los nodos del árbol de análisis en un analizador de abajo hacia arriba, para la entrada `program AABB;` primero, escribe el árbol y luego describe el orden.

9) Define gramática abstracta
10) Dibuja el AST para el siguiente fragmento de código Java:

```
a = b = f(a, b);
```

11) Dada la siguiente gramática ANTLR, dibuja el AST creado para la entrada: `- a > b + c`

```
expression returns [Expression ast]:
     ID                                  { $ast = new Variable($ID.text); }
    | e1=expression '>' e2=expression    { $ast = new Comparison($e1.ast, $e2.ast); }
    | e1=expression '+' e2=expression    { $ast = new Arithmetic($e1.ast, $e2.ast); }
    | '-' e1=expression                  { $ast = new UnaryMinus($e1.ast); }
    ;
```

12) ¿Cuál es la principal diferencia entre los parsers LL y LR?
13) ¿Cuál es la principal diferencia entre los parsers LL(1) y LL(*)?
14) ¿A qué hace referencia la primera L en los parsers LL y LR?
15) ¿Cómo se especifica la asociación de izquierda a derecha en ANTLR? Nombra dos operadores de asociaciones de derechas de un lenguaje conocido.
16) Añade código embebido a la siguiente gramática ANTLR para crear el AST más simple (obvia los número de fila y columna)

```
statement: 'write' expression (',' expression)*
         | ID '=' expression
         ;
```

17) Dada la asignación: `a[i+1] = record.field[a][b+2] = f(3);`
    a. Escribe una gramática BNF que reconozca este lenguaje
    b. Haz lo mismo con la notación EBNF
    c. Dibuja el árbol de análisis sintáctico de la gramática BNF
    d. Dibuja el árbol de análisis sintáctico de la gramática eBNF
    e. Dibuja el AST

18) Los lenguajes que soportan funciones como funciones de orden superior permiten las siguientes construcciones:
`myFunctionF(myFuctionG, 3) (myFunctionH)`
myFunctionF recibe otra función como primer parámetro y devuelve otra función que, a su vez, recibe otra función
    a. Escribe una gramática BNF que reconozca este tipo de expresiones
    b. Haz lo mismo con la notación EBNF
    c. Dibuja el AST de la expresión anterior
19) Dada la siguiente gramática:
`a → a a`
`a → ( a )`
`e → ε`
    a. Describe el lenguaje que genera
    b. Demuestra que es ambigua
20) Dada la siguiente gramática:

    exp → exp addop term
    exp → term
    addop → +
    addop → -
    term → term mulop factor
    term → factor
    mulop → *
    mulop → /
    factor → (exp)
    factor → INT_CONSTANT

    Escribe las derivaciones de un paso más a la izquierda, los árboles de análisis sintáctico y los AST para los siguientes programas:

    a. 3+4*5-6
    b. 3*(4-5+6)
    c. 3-(4+5*6)
21) ¿Es la gramática anterior ambigua? ¿Por qué?

Análisis semántico – Tema 4

Actividades y preguntas

1) Describe la semántica de la siguiente declaración: `fibonacci(v[i]);`
2) ¿Cuál es la principal diferencia entre la semántica del lenguaje y el análisis semántico?
3) Nombra 5 ejemplos diferentes de reglas semánticas en Java, proporcionando un ejemplo de cada una
4) Define gramática atribuída
5) Explica cuál es la evaluación de una gramática atribuída
6) Describe el algoritmo general para evaluar gramáticas atribuídas

7) Define una gramática de atributos bien definidos
8) Escribe un ejemplo de una gramática de atributos mal definidos
9) ¿Está bien definida la siguiente gramática?
   G:
      (1) Arithmetic: expression `1` `->` expression`2` + expression`3`
      (2) IntLiteral: expression `->` INT_CONSTANT
   R:
      (1) expression`1` .value = expression`2` .value + expression`3` .value
      (2) expression.type = new IntType()
10) ¿Se puede recorrer siempre cualquier gramática de atributos bien definidos con un solo recorrido lineal?
11) Define atributos heredados y sintetizados en una gramática atribuída
12) Extiende la siguiente gramática libre de contexto para convertirla en una gramática bien definida. En la gramática resultante, incluye un atributo heredado (llamado `inherited`) y otro sintetizado (`synthesized`)
   (1) nonTerminal1 `->` nonTermial2 TERMINAL1 nonTerminal3
   (2) nonTerminal2 `->` TERMINAL2
   (3) nonTerminal3 `->` TERMINAL3
13) ¿Qué es un recorrido válido para gramáticas de atributos bien definidos con todos los atributos sintetizados?
14) ¿Qué es un recorrido válido para gramáticas de atributos bien definidos con todos los atributos heredados?
15) ¿Qué es un recorrido válido para gramáticas de atributos bien definidos con todos los atributos sintetizados y heredados?
16) ¿Cuál es el mejor patrón de diseño para implementar la evaluación de una gramática atribuída simple?
17) En el Visitor:
    a. ¿Cuál es el nombre del método a implementar en los nodos del AST?
    b. ¿Cuál es el nombre del método a implementar en el Visitor particular?
    c. ¿Cuántos métodos con respecto a este patrón deben implementarse en un determinado visitor?
    d. ¿Cuántos métodos con respecto a este patrón deben implementarse en los AST nodes para `n` visitors?
    e. Dada una instancia de `TypeCheckingVisitor` y el nodo `Expression`, escribe el código para enlazar el visitor y la expresión
    f. ¿Cómo paso tres parámetros (`integer`, `double` and `String`) mientras atravieso un AST?
18) Dado el siguiente ejemplo:

```
double f(double r) {
    if (r<0) return -1;
    else return '0';
}
```

Para cualquier declaración de retorno, el tipo de su expresión debe ser un subtipo del tipo definido por la función. En el ejemplo anterior, int (-1) y char ('0') son subtipos de double, por lo que se cumple la condición.

Dada la siguiente gramática

(1) FuncDefinition: funcdefinition `->` type ID vardefinition* statement*
(2) FunctionType: type `->` type vardefinition*
(3) CharType: type `->` ε
(4) IntType: type `->` ε
(5) DoubleType: type `->` ε
(6) If: statement `1` `->` expression statement `2` + statement `3` *
(7) While: statement `1` `->` expression statement `2` *
(8) Return: statement `->` expression

Escriba una gramática de atributos para verificar que cualquier declaración de retorno realmente devuelva una expresión cuyo tipo es un subtipo del tipo declarado en la definición de la función. Puede asumir el diseño correcto del sistema de tipos y utilizar sus servicios (por ejemplo, métodos públicos).

19) ¿Cuál es el principal objetivo de la fase de Identificación?
20) ¿Cuál es el nombre de la estructura de datos usada en la fase de Identificación? ¿Cuál es su propósito?
21) ¿Cuáles son las dos fases del análisis semántico?
22) ¿Cuáles son las principales tareas en que consiste la Comprobación de Tipos?
23) Da 3 definiciones de tipo desde los puntos de vista denotacional, basado en la abstracción y constructivo.
24) Define:
    a. Type expression
    b. Type system
    c. Type checker
25) ¿Cuál es el propósito del tipo `Visitor` en el patrón de diseño Visitor?
26) Dada la expresión `f(v[i+1])`, traza todas las posibles operaciones que se pueden realizar en el analizador semántico
27) ¿Cuáles son los principales beneficios que aporta el patrón Composite para implementar un tipo system?
28) ¿Realiza la máquina virtual de Java algún análisis semántico cuando ejecuta los .class?
29) Escribe una gramática abstracta que convierta expresiones infijas a notación postfija
30) Dada la siguiente producción:
    a. Escribe una gramática atribuída para inferir el tipo de invocación
    b. Implementa un método del tipo sistema para inferir el tipo de invocación

Lenguajes Intermedios y Representaciones

Actividades y preguntas

1) ¿Por qué se usaron primero máquinas abstractas en la construcción de compiladores?
2) ¿Cuál es la principal diferencia entre una máquina abstracta y una virtual?
3) ¿En qué fases se usan comúnmente las representaciones de alto nivel? ¿Para qué se usan? Nombra 3 de ellas
4) ¿Qué es el middle-end de un compilador?
5) Nombra dos representaciones intermedias de medio nivel
6) Escribe en tres direcciones de código la siguiente expresión: `v[a+3*g].field`
7) Escribe el código MAPL para la siguiente expresión: `v[a+3*g].second` .Asume que todo son variables globales y:
   - La dirección de memoria de v es 0 y es un array de 10 elementos
   - Los elementos de v tienen dos campos: `first` (real con offset 0) y `second` (carácter con offset 4)
   - La dirección de memoria de `a` es 50 y es un carácter
   - La dirección de memoria de `g` es 51 y es un real
8) Escribe un programa de alto nivel donde la instrucción `dup` en MAPL es usada
9) Escribe un programa en MAPL equivalente al siguiente:

```
int i, j, n;
void p() {}
char c;
int f(int n, double r) {
  double real;
  char c1, c2;
  c1 = (char)n;
  real = (double)c1 + (double)n + r;
  p();
  return (int)c1;
}

void main() {
  struct {
    int integer;
    char character;
  } pair;
  int[10][5] vector;
  i=0;
  j=0;
  while(i<10){
    while(j<5) {
      vector[i][j] = i + j;
      j=j+1;
    }
    i=i+1;
  }
```

```
  i=0;
  pair.character = '0';
  pair.integer = 48;
  while (pair.integer >= i) {
    if (pair.integer == vector[0][0] || !(int)pair.character ||
i%2==0)
      write 't', 'r', 'u', 'e', '\n';
    else
      write 'f', 'a', 'l', 's', 'e', '\n';
    i=i+1;
  }
  write f(i, (double)i);
  vector[9][(int)4.3]=5;
  f(1, 2.2);
}
```

10) Pros y contras de las representaciones intermedias de bajo nivel
11) Escribe un diagrama de arquitectura de un procesador de lenguaje que use representaciones de leguaje de los 3 niveles de abstracción

Code Generation – Tema 6

Actividades y preguntas

1) ¿Cuál es el principal objetivo de la fase de generación de código?
2) ¿Cúales son los principales subproblemas que se deben abordar en la fase de generación de código?
3) ¿La generación de código es parte del front-end o del back-end de un compilador?
4) En MAPL, ¿cuál es la ecuación que define el offset de las variables locales?
5) En MAPL, ¿cuál es el principal propósito de BP?
6) En MAPL, ¿dónde apunta el BP?
7) ¿Qué funciones de generación de código deben definirse para la invocación de funciones?
8) Escribe dos ejemplos de código Java donde las asignaciones se usen como expresiones
9) Dada la siguiente función: `double f(char c, double d){ int[10] v; int w;};` En MAPL, ¿cuáles son las direcciones de memoria de c, d, v y w?
10) Detalla las partes de un marco de pila de funciones (registro de activación)
11) En MAPL, ¿cuál es la ecuación que define el offset de los parámetros de una función?
12) En Java, qué funciones de generación de código se deben definir para la operación de asignación
13) Dado `Type[size1][size2] w`; ¿Cuál es la dirección de memoria de `w[i][j]`?
14) Escribe la plantilla de código para acceder al campo de registro
15) Dado `int i; int j; double[10][20] w`; Escribe el código MAPL para la siguiente expresión: `w[i][j]`
16) Define la plantilla de código “execute” para la invocación de funciones para generar código MAPL
17) Dada la siguiente producción:
    `Assignment: expression 1 → expression2 expression3`
    Define la plantilla apropiada para generar su código MAPL

18) Escribe las plantillas necesarias para generar el código MAPL de un Switch
19) Escribe las plantillas necesarias para generar el código x86 de las siguientes instrucciones MAPL: `push`, `add`, `load` y `store`.
20) Considera el operador '+=' como un nuevo tipo de statement (no expresión) para un lenguaje determinado. Con tal operador, las siguientes declaraciones aceptadas por el compilador son:

```
myInteger += 1;
myDouble += myDouble * 2.1;
myIntVector[myInteger+1] += myInteger*2;
myRecord.intField += function_returning_int(myInteger)
```

Los tipos de los dos operandos deben ser iguales y deben ser enteros o dobles. No se permiten caracteres.

   a. Escribe la producción abstracta (no concreta) que especifica el nuevo += statement
   b. Identifica las funciones de código necesarias para la producción previa y escribe las correspondientes plantillas de código para la producción anterior

21) Escribe las plantillas de código necesarias para generar código MAPL para las operaciones aritmética(+, -, * y /) y comparación (>, <, >=, <=, == y !=) para cualquier combinación de expresiones char, int y double. Ambas operaciones permiten operandos de cualquiera de esos 3 tipos