CC3006: Tarea #1

Para entregar el Lunes, Enero 25, 2010

Bidkar Pojoy

Carlos E. López Camey

Problema 1

a) Cuál es la finalidad de los compiladores?

Traducir un lenguaje de programación L_1, que es un lenguaje que entendemos en cierta manera los humanos, a otro lenguaje L_2. Usualmente, L_2 es código de más "bajo nivel", es decir, la mayoría de veces, código objeto. El código objeto es, típicamente, código de máquina o instrucciones binarias.

El proceso de traducción o compilar se hace con el objetivo de crear un programa *ejecutable.*

b) Qué tienen en común los compiladores y los intérpretes?

Los dos actúan en algún momento como traductores.

c) Qué diferencia existe entre un compilador y un intérprete?

El compilador produce un código traducido a partir de un lenguaje de programación, generalmente en un archivo. Un intérprete, además de traducir (generalmente), interpreta el lenguaje de programación, como su nombre lo dice.

Problema 2

a) Enumere lenguajes de programación que estén implementados como compiladores.

Pascal, C, C++, Delphi.

b) Enumere lenguajes de programación que estén implementados como intérpretes

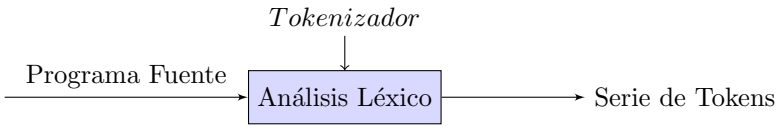
Ruby, Python, Perl, Lisp.

Problema 3

Describa las fases de un compilador junto al flujo de información que existe entre ellas.

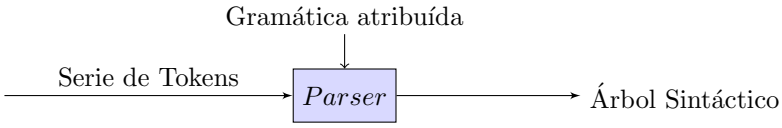
- **Análisis Léxico**: Convierte o separa el programa desde una serie de caracteres, a una serie de *tokens* llamados **Lexemas** que representan los bloques del programa. Los *tokens* son palabras "clave" del lenguaje i.e. palabras como *if,print,while*, operadores matemáticos como +,*, etc.

Los *tokens* generados son de la forma $< nombre - del - token, valor - atribuído >$, que están representados en una **Tabla de Símbolos**, similar a una $hash - table$.



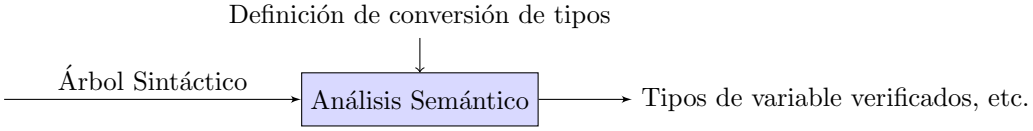
- **Análisis Sintáctico o *Parseo***: Chequea que los *tokens* recibidos a partir del análisis léxico aparezcan en el orden correcto, es decir, que formen instrucciones gramáticalmente correctas. Por ejemplo, en Java, la instrucción $variable++$; está sintácticamente correcta, pero la instrucción $++variable$; no, el orden de los *tokens* está mal.

  En esta fase, se produce el llamado **Árbol Sintáctico** (o una estructura similar), representando la estructura gramatical de los *tokens* recibidos. Para poder generar el árbol sintáctico, se necesita una definición de una **Gramática**.



- **Análisis Semántico**: Se analiza el $significado$ del árbol semántico construído. Esto se refiere a los aspectos tales como chequear que las variables utilizadas hayan sido declaradas e.g. $variable++$; puede estar gramaticalmente-correcta pero si la variable *variable* no fue declarada, entonces esta semánticamente incorrecta.

  Opcionalmente, se puede requerir de una definición de conversión de tipos ímplicita, conocida como ***Coercion*** (en inglés) para la verificación de tipos de variable e.g. una variable *entera* se podría operar con una *booleana*. Algunos lenguajes "dejan" que los compiladores puedan definir esta conversión de tipos. Además de verificar los tipos de las variables con las que pueden operar, se verifica que se haya salido de un *ciclo*, se analizaa el ámbito de las variables, etc.



- **Generación de Código**: Se traduce el programa fuente al código objetivo, en donde se pueden construír más de una representación intermedia, con diferentes formas.

Luego de la generación de código, el compilador puede pasar por una pase de optimización de código.

Problema 4

Qué son las context-free-grammars o grámaticas independientes del contexto?

Es una definición de una gramática G (utilizada en el análisis sintáctico), que consta de:

1. Un conjunto de **terminales** (los *tokens* generados por el análisis léxico).
2. Un conjunto de **no terminales** (variables sinácticas)

3. Un conjunto de cadenas (Reglas o producciones posibles)

4. Una definición de un **no terminal** como símbolo inicial de G.

Ejemplo, G:
$L- > L$ '+' d
$L- > L$ '-' d
$L- > d$
$L- >' 0'||'1'||'2'...'9'$

En donde el símbolo inicial es el **no terminal** L.

Definimos aquí a $L(G) = \{w|w$ es derivada a partir de $G\}$, que es el conjunto de todas las producciones posibles a partir de G

Definimos también al símbolo ϵ como la cadena de cero terminales (o no terminales) i.e. una cadena vacía.

Una producción posible es $w = 5$ '+' 1, que proviene a partir de las reglas producidas:

$L- > L$ '+' d (Regla 1)
$L- > d$ '+' d (Regla 3)
$L- > 5$ '+' d (Regla 4)
$L- > 5$ '+' 1 (Regla 4)

Y decimos que $w \in L(G)$

Problema 5

Que significa ambiguedad en las gramáticas independientes del contexto. Ejemplifique

Dada una gramática G, decimos que esta es ambigua si y solo sí, una producción $w \in L(G)$, puede ser 'generada' o inferida por dos series de reglas diferentes.

Ejemplo: Dada la gramática G: $S- > S$ '+' S
$S- > S$ '-' S
$S- >' 0'||'1'||'2'...'9'$

Verifiquemos $w = 9$ '-' 5 '+' $2 \in L(G)$

Inferencia 1:
$S- > S$ '-' S (Regla 2)
$S- > 9$ '-' S (Regla 3)
$S- > 9$ '-' S '+' S (Regla 1)
$S- > 9$ '-' 5 '+' S (Regla 3)
$S- > 9$ '-' 5 '+' 2 (Regla 3)

Inferencia 2:
$S- > S$ '+' S (Regla 1)
$S- > S$ '-' S '+' S (Regla 2)
$S- > 9$ '-' S '+' S (Regla 3)
$S- > 9$ '-' 5 '+' S (Regla 3)

$S->$ 9 '-' 5 '+' 2 (Regla 3)

Claramente, vemos que w pudo ser inferida de dos diferentes maneras, entonces decimos que G es ambigua.

Problema 6

Defina que es Árbol Sintáctico

Sea G una gramática independiente del contexto, un árbol sintáctico cumple con:

1. La raíz del árbol es un símbolo gramatical $\in G$ no terminal
2. Las hojas (nodos sin hijos), son símbolos gramaticales $\in G$ terminales o ϵ
3. Cada nodo interior, es un símbolo gramatical $\in G$ no terminal
4. Para todo nodo interior A, cuyos hijos sean $X_1, X_2 \dots X_n$ de izquierda a derecha, entonces existe una producción $A = X_1 X_2 \dots X_n \in L(G)$. Aquí $X_1, X_2 \dots X_n$ pueden ser símbolos gramaticales terminales o no terminales.

Problema 7

Cuál es la diferencia entre un componente léxico (*token*) y un lexema?

Un **lexema** es una secuencia de caracteres en el programa fuente que *matchea* (concuerda) un patrón para un *token* y es identificado por el analizador léxico o *scanner* como una instancia de ese *token*. Un patrón es una descripción de la forma que los lexemas de un token puedan tomar.

Por ejemplo, el lexema **24010** es una instancia del token *numero*, por que concuerda (hace *match*) con la descripción del token < *numero*,cualquier constante numérica>

Problema 8

Cuales son los elementos principales de una gramática independiente del contexto?

Los elementos importantes, son los 3 conjuntos del que consta la gramática, y la especificación del símbolo inicial.

Eso es,

1. Un conjunto de **terminales** (los *tokens* generados por el análisis léxico).
2. Un conjunto de **no terminales** (variables sinácticas)
3. Un conjunto de cadenas (Reglas o producciones posibles)
4. Una definición de un **no terminal** como símbolo inicial de G.

Problema 9

Dada la Gramática G:

$S -> SS$ '+'
$S -> SS$ '*'
$S -> a$

Donde S es el símbolo inicial y $a, +$ y * son terminales para la gramática. Demuestre que la cadena "aa+a*" pertenece a $L(G)$

Demostración:
$S -> SS$ '*' (Regla 2)
$S -> SS$ '+' S '*' (Regla 1)
$S -> aS$ '+' S '*' (Regla 3)
$S -> aa$ '+' S '*' (Regla 3)
$S -> aa$ '+' a '*' (Regla 3)
□

Problema 10

Defina que es traducción dirigida por la sintaxis.

Es una traducción que se hace adjutando reglas o fragmentos de programa a una producción en una gramática. Por ejemplo, podríamos tener la siguiente producción para el no-terminal *exp*

$$exp -> exp_1 + \textit{otro-término}$$

podríamos llegar a traducir *exp* como la suma de dos sub-expresiones (exp_1 y *otro-término*), en pseudo-código, algo como:

traducir exp_1

traducir *otro-término*

manejar +

Entonces, se asocia un elemento gramatical con un atributo, y para cada producción, asociamos un conjunto de reglas semánticas i.e. instrucciones para calcular los atributos. Un ejemplo de un atributo podría ser, el tipo de dato de una expresión.

Problema Extra

Dada la gramática G

$num -> 11$
$num -> 1001$
$num -> num\ 0$
$num -> num\ num$

Demuestre que todas las producciones $w \in L(G)$ son divisibles por 3.

Solución:

Notemos que, la **Regla #1** y la **Regla #2** dan ya a números divisibles por 3, eso es que '11_b' es 3 en decimal y '1001_b' es 9.

Sabemos que para convertir de binario a decimal, usamos el 2 como base y lo elevamos a la potencia correspondiente (de la casilla, empezando desde 0 y la derecha) dependiendo si el bit del número binario en la casilla es un '1' o un '0'. Por ejemplo, vemos que $11_b = 2^0 + 2^1$, $1001_b = 2^0 + 2^3 = 9$

Supongamos que en la Regla #3, *num* es un múltiplo de 3, eso es $num = 3x$, que es cierto si *num* 'proviene' de la Regla #1 y #2. Si anadimos un bit '0' a la derecha (como dice la Regla que queremos probar), notemos que, con los ejemplos

$$\text{'11'} + \text{'0'} = \text{'110'} = 2^1 + 2^2 = 6$$

Que es el mismo resultado de, sumarle 1 a los exponentes de las bases sumadas de *num* anterior. Eso es, en vez de 2^0 y 2^1, que suman 3, ahora tenemos 2^1 y 2^2. Al anadir el '0' a la derecha, estamos básicamente mutiplicando por 2^1 cada número que estamos sumando. En efecto,

$$num_b = 2^0 + 2^1 \dots 2^n$$

$$num_b + \text{'0'} = (2^0 + 2^1 \dots 2^n) * (2^1)$$

$$num_b + \text{'0'} = 2^{0+1} + 2^{1+1} \dots 2^{n+1}$$

e.g. '11' + '0' = '110' = $(2^0 + 2^1) * (2^1) = (2^1 + 2^2) = 6$

Si estamos multiplicando a *num* -que es un supuesto múltiplo de 3- por dos, tenemos $3x*2$ que es también múltiplo de 3. Probamos que la Regla 3 produce números múltiplos de 3, si y solo si, al número al que le estamos anadiendo el '0' por la derecha, es múltiplo de 3.

Para probar la **Regla #4**, re-escribamos $num- > num_1\ num_2$, para diferenciar. Notemos que si num_1 y num_2 son múltiplos de 3, tenemos, $num_1 = 3 * u$ y $num_2 = 3 * v$. La Regla #4 dice básicamente que la suma de dos *num* (diferentes o iguales), es otro *num*. Pero num_1 está en cierta forma 'desplazado' hacia la izquierda n posiciones, es decir, como en la Regla #3, multiplicar por algun múltiplo de 2 i.e. 2^n. Entonces tenemos $num = num_1 + num_2 = 3 * u * 2^n + 3 * v = 3 * (2^n * u + v)$ que también es un múltiplo de 3.

"Probando" con el "caso base", es decir, con la Regla #1 y #2, tenemos $num- > num + num =$ '11' + '1001' = $(2^0 + 2^1) * 2^4 + (2^0 + 2^3) = 3 * 2^4 + 9 = 57 =$ '111001_b' □