Participante:
Ortega Bustamante Antonio

No. Control:
171080148

Escuela:
Instituto tecnológico de Iztapalapa

Materia:
Lenguajes y autómatas II

Tarea:
Actualización de apuntes

NOTA:
Este trabajo se realizó de manera individual por Antonio ortega Bustamante

Mod-01 Lec-01

Los compiladores están por todas partes ya que hay muchas aplicaciones para la tecnología de compiladores., Se encuentran desde los siguientes tipos:

-intérpretes para javascript/flash

- maquinas generadoras de código para niveles de lenguajes

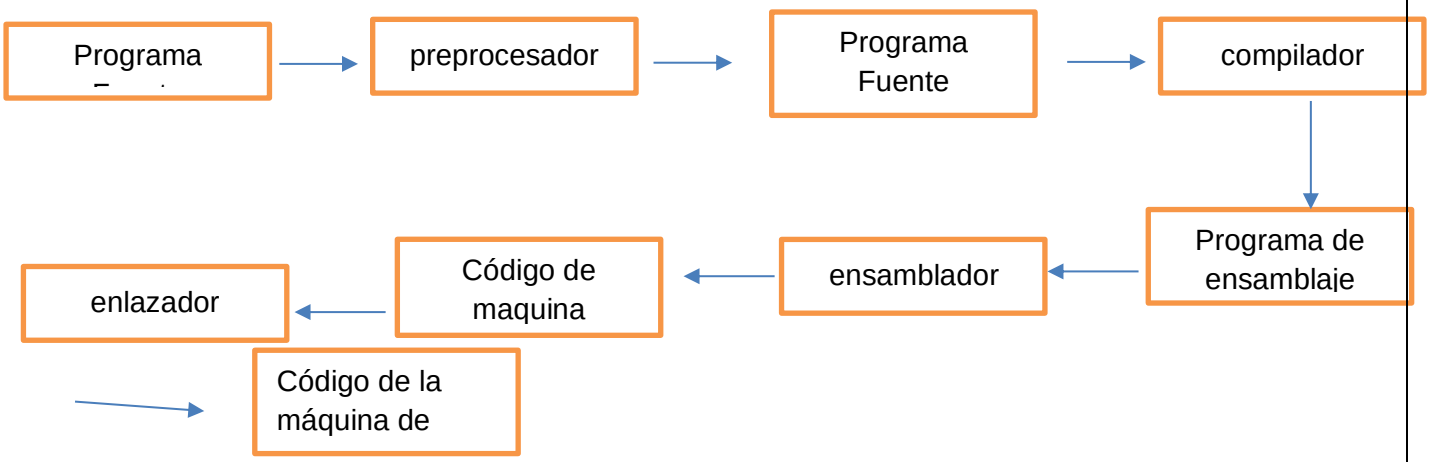
-optimización de programas

- detección de código malicioso

Entre otros

Un compilador es posiblemente el software de sistema más complejo y escribirlo es un ejercicio importante en ingeniería de software. Ya que su complejidad surge del hecho de que es necesario mapear los requerimientos de un programador (in un programa HLL) a los detalles arquitectónicos.

Estos requieren algoritmos y técnicas de un gran número de áreas en la ciencia computacional, pasando de la teoría a la práctica y se estructura de la siguiente manera el sistema de procesamiento de lenguaje



¿Qué es el análisis léxico?

Es la entrada de un programa de lenguaje de alto nivel, como un programa 'c' en forma de una secuencia de caracteres, también es la salida es una secuencia de tokens que se envía al analizador para el análisis de sintaxis.

Elimina los espacios en blanco, las pestañas, las nuevas líneas y los comentarios del programa de origen y realiza un seguimiento de la línea, números y asocia los mensajes de error de varias partes de un compilador con línea

Tokens

Es una cadena de caracteres que lógicamente pertenecen juntos, por ejemplo:

flotante, identificador, igual, menos, suma, punto y coma, etc.

Pattern

Es el conjunto de cadenas para las que se produce el mismo token, el patrón se dice que coincide con cada cadena en el conjunto

Lexema

Son la secuencia de caracteres relacionados por el patrón que corresponde al token, por ejemplo “float”, “abs_zero_Kelvin”, “=”, “273”, etc.

Mod-02 lec-04 lexical analysis

Diagramas DFA

Los diagramas de transición son DFA generalizadas con las siguientes diferencias diagramas de transición son DFA generalizadas con las siguientes diferencias

- bordes pueden estar etiquetados por un símbolo,
- un conjunto de símbolos o una definición regular, algunos estados que aceptan pueden indicarse como estados retraídos, lo que indica que el lexema no incluye el símbolo

- cada mirada de aceptación tiene una acción adjunta que se ejecuta cuando se alcanza ese estado

Los diagramas de transición no están destinados a la traducción automática, sino sólo para la traducción manual

lex- un generador analizador léxico

lex tiene un lenguaje para describir expresiones regulares
genera un emparejador de patrones para las especificaciones de expresión regular que se le proporcionan como entrada

Mood-03 Lec-06 Syntax Analysis

Al igual que en el caso de NFA y DFA, PDA también tiene dos versiones: NPDA y DPDA sin embargo NPDA son estrictamente más poderosos que el DPDA en la práctica necesitamos DPDA ya que tienen exactamente un posible movimiento en cualquier instante.

Parsing

el análisis es el proceso de construir un árbol de análisis para una oración generada por una gramática dada, si no hay restricciones en el idioma y la forma de gramática utilizada, los analizadores para los idiomas sin contexto requieren
análisis de arriba hacia abajo mediante el análisis predictivo traza la derivación más a la izquierda de la cadena mientras se construye el árbol de análisis estrellas del símbolo estrella de la gramática, y predice la próxima producción utilizada en la derivación tal predicción se ayuda por el análisis de tablas
la siguiente producción que se utilizará en la derivación se determina utilizando el símbolo de entrada nets para buscar la tabla de análisis (símbolo de búsqueda anticipada) la colocación de restricciones a la gramática garantiza que ninguna ranura en la tabla de análisis contiene más de una producción en el momento de analizar la construcción de la tabla, si dos producciones pasan a ser elegibles para ser colocadas en la misma ranura de la tabla de análisis.

Mood-03 Lec-07 Syntax Analysis

una gramática recursiva a la izquierda tiene un no terminal de tal manera que A-> Aa los métodos de análisis de arriba hacia abajo (LL(1) y RD) no pueden manejar gramáticas recursivas la recursión izquierda en las gramáticas puede ser eliminada por transformaciones.

mango: un mango de una forma sentencial derecha es una producción A--> & y la posición en la que la cadena puede ser encontrada y reemplazada por A para producir la forma de sentencial derecha anterior en una derivación más a la derecha de que si S-> mm un AW > aBW.

Un mango siempre aparecerá finalmente en la parte superior de la pila nunca sumergido dentro de la pila en el análisis S-R localizamos el mango y lo reducimos por el LHS de la producción repetidamente para llegar al símbolo de inicio

una configuración de un LR es (s0 X1 s2 X2... Xm Sm a!a!a!+1....An $)

configuración inicial del analizador donde s0 en el estado inicial del analizador y (a1a2... an$) es la cadena que se va a analizar dos partes en la tabla de análisis ACTION y GOTO 1) la tabla ACTION puede tener cuatro tipos de Shift entripado, reducir, aceptar o error

2) la tabla GOTO proporciona la siguiente información de estado que se utilizará después de un movimiento de reducción

considerar una derivación más a la derecha

S->rm Bt->rm

donde se ha aplicado la producción B --> una gramática se dice que es LR(k) si para cualquier cadena de entrada dada, en cada paso examinando la cadena y escaneando como máximo los primeros símbolos k de la cadena de entrada no utilizada.

teorema el conjunto de todos los prefijos viables de todas las formas sentenciales correctas de una gramática es un lenguaje regular el DFA de este lenguaje regular puede detectar manejadores durante el análisis cuando este DFA alcanza un "estado de reducción", el prefijo viable correspondiente no puede crecer más y, por lo tanto, indica una este DFA puede ser construido por el compilador usando la gramática todos los analizadores LR tienen tal DFA incorporado en ellos

Mod-0- Lec 10 Syntax Analysis

cambiar y reducir las acciones si un estado contiene un elemento del formulario (A --> a.) entonces una reducción por la producción A --> es la acción en ese estado si no hay elementos de reducción en un estado, entonces el cambio es la acción apropiada podrían ser cambios, reducir conflictos o reducir, reducir los conflictos en un estado si no hay conflictos s-S-R o R-R en cualquier estado de un DFA LR (0), entonces la gramática es LR (0)

Los elementos LR (1) tienen la forma [A--> n,b,a] siendo el símbolo de búsqueda los símbolos de la mira no tienen ningún papel que desempeñar en los elementos de turno, pero en reducir los elementos de la forma [A--> un... a] reducción por A--> a es válida sólo si el siguiente símbolo de entrada es "a"

Mod 04 Lec- 12 semántica análisis

la consistencia semántica que no se puede manejar en la etapa de análisis se maneja aquí, Los analizadores no pueden manejar las características de contexto-sensitivo de los lenguajes de programación estos son semántica estática de lenguajes de programación y pueden ser comprobados por el analizador semántico

- las variables se declaran antes de su uso
- tipos coinciden en ambos lados de las asignaciones
- tipos de parámetros y número coinciden en declaración y uso

compiladores sólo pueden generar código para comprobar la semántica dinámica de los lenguajes de programación en tiempo de ejecución

- si se producirá un desbordamiento durante una operación aritmética
- si los límites de la matriz se cruzarán durante la ejecución
- si la recursividad cruzará los límites de la pila
- si la memoria del montón será insuficiente

```
Int dot_prod( int x[], int y[]); {
```

```
Int d, i; d = 0 ;
For (i=0; i<10; i++) d += x [i]* y[i];
Return d;
}
Main(){
Int p; int a[10], b[10];
P = dot_prod (a,b);
```

Examen Corregido

Ejercicio 11 cap. 2

el algoritmo de minimización DFA dado en figura 2.9 está formulado para enumerar todos los elementos de P y todos los caracteres en cada iteración del bucle while

a) **refundir el algoritmo de modo que utilice un pool de trabajo para contener los conjuntos que todavía deben ser examinados**
b) **b) refundir la función de división para que particione el conjunto alrededor de todos los caracteres en E**

T ← {DA, { D − DA} };
P ← ∅
while (P 6= T) do
P ← T;
T ← ∅;
for each set p ∈ P do
T ← T ∪ Split(p);
end;
 	
end;
Split(S) {
for each c ∈ 6 do
if c splits S into s1 and s2
then return {s1
,s2
};
end;
return S;
	T ← {DA, { D − DA} };
	P ← ∅
	while (P 6= T) do
	P ← T;
	T ← ∅;
	for each set p ∈ P do
	T ← T ∪ Split(p);
	end;
	
	while (estado 1=3)
	{
	Simbolo= get Simbolo ();
	Estado= T [estado] [Simbolo]
	}
	Split(S) {
	for each c ∈ 6 do
	if c splits S into s1 and s2
	then return {s1
	,s2
	};
	end;
	return S;

c) **¿cómo la complejidad del caso esperado c lenguaje construye algoritmo original**

```
T→ DA (D/ DA){
P → 0
While (P 6=DA)
P → 0
T =/ P
For ( p + T; t+ p; p++)
T= P;
}
```

ejercicio 14 cap. 3

Considere la tarea de crear un analizador para el lenguaje de programación Esquema.

Contraste el esfuerzo necesario para un descenso recursivo de arriba hacia abajo analizador con el necesario para un analizador Lr (1) controlado por tabla.

(Supongamos que ya tiene un generador de tablas Lr (1).)

Descenso de LR1
LR → L+R
LR → L*R
LR → L-R
LR → L+L
LR → L+L*R
LR → L+L*R-L
LR → R*R
LR → R-R*L
LR → R-R*L-L