Teoría de Lenguajes, Autómatas y Compiladores

Trabajo Práctico Especial 2 - YACC

Matias Domingues - Legajo 50278

Objetivo

Desarrollo de un compilador que, a partir de una gramática dada, genere código en C.

Gramática

Se define la gramática G = (Vn, Vt, S, P) donde:
Vn = {NAME EQQ SUMM NUMBER NEQ RETURN LESS MILL VARTYPE IF WHILE PRINT}

Vt={0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z, VOID, (,), ++, --, >, <, ==, !=, =, +, -, / *, %, int, char, while, if, return}

S = Símbolo Inicial

P = Producciones

Desarrollo

Para la realización de este proyecto, se decidió utilizar una gramática correspondiente a un lenguaje imperativo, muy similar a C, aunque con variaciones enumeradas a continuación.

- Se implementó el símbolo &int el cual devuelve un 1 si el número es primo, o 0 en caso contrario.
- Se implementó el símbolo print, al cual se le pasa como argumento una variable a imprimir en pantalla.
- Se implementó el símbolo getc, el cual funciona de la misma manera que getchar() de C.
- Se implementaron las sentencias if y while.
- Se acepta recursión y llamadas a funciones externas.
- Se aceptan variables del tipo int y char.

Durante el desarrollo, noté que utilizar el analizador léxico para validar el input era ineficaz, ya que era más simple realizarlo en el analizador semántico. Esto generó una gramática final más simple.

Cálculo de primos

Se utilizó el método Miller Rabin para realizar el test de primalidad. Es similar a la prueba del teorema de Fermat, pero tiene mayor probabilidad de asegurar si un número es primo o no.
El algoritmo al ingresar un número impar n selecciona un número al azar a entre $1 \leq a \leq n-1$

Finalmente, si $a^s \equiv 1 \pmod{n}$ o $a^{2^j s} \equiv -1 \pmod{n}$ se considera que n pasa la prueba.
Para tener mayor efectividad se hace la prueba con distintos valores de j, ya que un número primo tendría que pasar la prueba con cualquiera. Para aumentar la efectividad en el algoritmo implementado, se hacen varias pruebas utilizando los primeros 7 números primos, ya que esto es útil para el cálculo de un primo menos a 3.4e10.

Futuras extensiones

Se podría implementar la extensión para Big Integer, ya que por el momento solo soporta uint32_t y no hay una ganancia visible para poder implementar uint64_t.
Se podría implementar un léxico más completo para soportar distintos tipos de operadores y tipos de variables. Se encontraba planeado implementar un tipo boolean.
Se encontraba planeado implementar bloques (del tipo Smalltalk) pero no era necesario para el calculo de numeros primos.

Referencias

Para no realizar el algoritmo nuevamente, se copio el metodo de un repositorio publico de Github: https://github.com/rflynn/euler/blob/master/miller-rabin.c