Compilador MFJG

Martin Purita, Legajo 51187
Facundo Menzella, Legajo 51533
Julián Gutiérrez Ferrara, Legajo 51141
Gustavo Del Guidice, Legajo 51289

Objetivo

Desarrollar un compilador que sea capaz de factorizar un número grande en sus factores primos.

El Problema

Al avanzar con el desarrollo de la segunda parte, nos dimos cuenta que la gramática que habíamos definido inicialmente validaba muchas cosas que seria mas cómodo verificar en la semántica, Esto se debió a que todavía no poseíamos los conocimientos sobre la estructura semántica al momento de confeccionar la gramática

La solución

Se realizo un cambio drástico en la misma, adicionalmente y en consideración del objetivo de nuestro compilador, se simplifico de forma tal que solo admita una función principal, y una lista de números a factorizar en primos por medio de la palabra reservada "*decompose*" de nuestra sintaxis.

La gramática

La nueva gramática se define de la siguiente forma

G = (Vn, Vt, S, P) donde:

Vn = {S, LEFT_PARENTHESIS, RIGHT_PARENTHESIS, LEFT_BRACE , RIGHT_BRACE , SEMI_COLON, RETURN, DECOMPOSE, TYPE,CONST,ID};

Vt = {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,0,1,2,3,4,5,6,7,8,9,{,},(,)};

S símbolo inicial;

P producciones;

La semántica

Como se mencionó anteriormente, se paso gran parte de las validaciones de la gramática aquí, internamente al compilar se ejecuta la descomposición en primos y queda en a.out un archivo de c con una lista de printfs con los resultados. Esta es la representación en C de un programa de nuestra sintaxis, elegimos este método porque es mas sencillo que soportar salida estándar, y mucho mas eficiente a la hora de ejecutarse mas de una vez. El método de entrada debe ser manual por medio del programa a compilar, ya que no soportamos entrada estándar.

Detrás del telón

Cada vez que se invoca a la palabra reservada "*decompose*" con un numero primo como parámetro, durante la compilación se ejecuta la descomposición en primos, tomando en numero de entrada como valor entero de 64bits convertido mediante la función *atoi* de C y el resultado se vuelva en un printf de c directamente dentro de la cadena.

Ref. Externas

En busca de una forma eficiente de descomponer números primos se utilizo una implementación con cache[1] , la primera ejecución genera un archivo adicional primebits de alrededor de 137mb con todos los números primos que entran en un int de 64 bits, por lo que la primer ejecución toma alrededor de 1 minuto, y luego se reduce a 1 segundo. El algoritmo se basa en una criba (como la criba de Eratóstenes[2]) pero utilizando una rueda de primos[3] de tamaño 30

Restricciones de tiempo

Como siempre, al encarar un proyecto, una de las grandes limitantes es el tiempo. Nuestra prioridad fue administrar el mismo para que el compilador logre, lo mejor posible , su objetivo.

Fue por eso que concentramos nuestras energías en hacer funcionar una implementación veloz para descomponer primos y en darle foco a la sintaxis para soportarla en forma sencilla, de modo tal que una persona sin demasiados conocimientos de programación, pueda obtener descomposiciones en forma rápida mediante nuestro compilador especializado.

Pendientes

Como mejoras al TPE seria bueno soportar BIG INTEGER como parámetro en la descomposición, ya que por ahora solo soportamos int de 64bits, adicionalmente seria bueno también que el código equivalente a nuestra sintaxis sea en ASM en vez de en C, para mayor velocidad.

Adicionalmente si le diera algún tipo de salida a la palabra reservada en C se podrían definir operaciones aritméticas básicas sobre ese resultado (suma, resta multiplicación y división).

Esto invitaría a la gramática a soportar recursividad y variables para darle mas poder realizar operaciones mas interesantes desde el punto de vista matemático y darle mas potencia a nuestro lenguaje.

Una de las razones por las que no se hizo esto, es porque habría que definir las operaciones aritméticas para una descomposición en primos, habría que manejarlas internamente en el caso de soportar BIG INTEGER y habría que darle una forma de salida estándar a la gramática.

Código fuente

El código fuente se puede ver en el repositorio de GitHub: https://github.com/MartuPuri/TLA/ encontraran también un archivo de README con el modo de uso del mismo.

[1] http://rosettacode.org/wiki/Prime_decomposition#Using_GNU_Compiler_Collection_gcc_extensions

[2] http://es.wikipedia.org/wiki/Criba_de_Erat%C3%B3stenes

[3] http://primes.utm.edu/glossary/xpage/WheelFactorization.html