Instituto Tecnológico de Buenos Aires
Teoría de Lenguajes y Autómatas

Trabajo Práctico 1:

Analizador Léxico Genómico

Integrantes:
Altamiranda Enzo, 51265
Camisay Santiago, 47583
Costesich Pablo, 50109

Índice:

Introducción 2

Instrucciones 3

Decisiones de Diseño 3

Descripción de la Gramática 5

Dificultades Encontradas 6

Futuras Extensiones 6

Introducción:

En el siguiente informe se explicará brevemente el uso del programa para analizar secuencias genómicas, así como también las decisiones de diseño más importantes que se utilizaron para su creación, la secuencia genómica que genera cadenas permitidas por el programa, algunos problemas encontrados en la creación del mismo y qué mejoras se le podría hacer para mejorar su rendimiento.

Instrucciones:

En la carpeta 'root' existe el directorio 'src'. En el mismo se encuentra el archivo *syntax.l* en donde está el código del programa. En la raíz del directorio existe el archivo *test_file.sh* el cual contiene varias pruebas del programa. Al ejecutarse compila el programa y automáticamente corre varios tests para probar el correcto funcionamiento del programa. Para compilarlo, se utiliza un archivo *makefile* y los tests más importantes se encuentran en el directorio 'examples'.

Dentro del directorio 'examples', se encuentran archivos con cadenas genómicas. Enre los mismos se pueden encontrar las cadenas provistas por la cátedra, así como también otras cadenas obtenidas de una fuente de internet[1].

Al ejecutar el archivo *test_file.sh*, se obtiene en el *stdout* las cadenas de aminoácidos traducidas a partir de las cadenas genómicas.

Decisiones de Diseño:

El programa intenta imitar de forma rudimentaria el diseño de una máquina de estados, donde se analizan la combinación de caracteres que podrían ocurrir en una secuencia genómica válida según el reglamento de la cátedra. Cada caso se podría considerar como un estado y dado ese estado sólo existe un conjunto de posibles caracteres que el programa espera encontrar. Si una secuencia de caracteres no concuerda con lo esperado, entonces como en una máquina de estados, donde se encuentra una entrada que no lleva a ningún estado válido, la cadena es rechazada.

La cadena es leída de a un carácter por vez y el mismo es almacenado en un buffer. Cuando este es completado con tres caracteres, se procede a verificar qué cacteres son y a mostrar el aminoácido correspondiente en el *stdout* dado el caso de que sea una combinación válida.

Para realizar la codificación de cadena de tres genomas a aminoácidos, se pensó en utilizar la estructura de datos llamada *trie*. El mismo tendría en cada en cada una de sus hojas el aminoácido que corresponde a la secuencia que se siguió desde la raíz hasta la misma. Cada nodo tendría un arreglo de punteros a nodos hijo, en donde habría una relación directa entre una letra del abecedario y una posición del arreglo. Bajo estas condiciones la complejidad de la búsqueda siempre sería **f(n) = 3**, donde **n** es algún lexema válido, ya que sólo habría que recorrer tres nodos antes de llegar a la hoja y conseguir el aminoácido correspondiente al lexema.

[1] Parte de la secuencia genómica de una gallina:
http://www.ncbi.nlm.nih.gov/nuccore/J04598.1

Posteriormente, se decidió utilizar una tabla de codificación de lexemas[2], en donde algunos lexemas quedarían codificados en sólo uno o dos pasos. Si bien ahorarse un paso puede parecer trivial, cuando se tenga que analizar cadenas de longitudes grandes, el tiempo ahorrado por no tener que completar algunos pasos puede resultar en una diferencia significativa.

[2] La misma se puede encontrar en:
http://en.wikipedia.org/wiki/DNA_codon_table

Descripción de la Gramática:

Se creó la siguiente gramática para representar las secuencias genómicas aceptadas por el programa:

G = < VN , VT , P , <START> >

VN = { <START> , <HEADER> , <BODY> , <END> , <CODON> , <POSITION> , <DISJUNCTION> , <BASE> , <DIGIT> , <D> , <BASE_STREAM> , <DISJUNCTION_PART> , <DISJUNCTION_OPTIONAL> }

VT = { A , U , G , T , S , O , P , - , 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , [,] , { , } , * }

P = {
<START> -> <HEADER> <BODY> <END>
,
<HEADER> -> AUG
,
<END> -> STOP | TAA | TAG | TGA
,
<BODY> -> <CODON> <BODY> | <POSITION> <BODY> | <DISJUNCTION> <BODY> | λ
,
<CODON> -> <BASE><BASE><BASE>
,
<BASE> -> C | G | A | T | -
,
<POSITION> -> [<DIGIT>]
,
<DIGIT> -> 1 <D> | 2 <D> | 3 <D> | 4 <D> | 5 <D> | 6 <D> | 7 <D> | 8 <D> | 9 <D>
,
<D> -> 0 <D> | 1 <D> | 2 <D> | 3 <D> | 4 <D> | 5 <D> | 6 <D> | 7 <D> | 8 <D> | 9 <D> | λ
,
<DISJUNCTION> -> { <BASE_STREAM> <DISJUNCTION_PART> } <DISJUNCTION_OPTIONAL>
,
<DISJUNCTION_PART> -> ,<BASE_STREAM> <DISJUNCTION_PART>
,
<DISJUNCTION_OPTIONAL> -> * | λ
,
<BASE_STREAM> -> <BASE> <BASE_STREAM> | λ
}

Dificultades Encontradas:

En la creación del programa se encontró sólo una dificultad técnica. Se quizo modularizar el código fuente en varios archivos, ya que sería conveniente que en archivo de *lex* quede sólo el código necesario para hacer el parseo. Sin embargo, no se pudo hacer que el archivo de *lex*, *syntax.l*, reconociera variables globales declaradas en otros módulos y ya que en la escritura del código se usaron exténsamente estas variables era impráctico rediseñar el programa. El grupo debe averiguar a qué se debe esta limitación de no poder importar variables globales de otros módulos.

Futuras Extensiones:

Actualmente el programa utiliza un buffer en donde se almacenan tres caracteres, que luego se verifican para saber si hay correspondencia entre la secuencia y algún codón. Sería más conveniente que dentro de las reglas de parseo que utiliza *lex*, esté contempado el análisis de estos codones. De tal forma que al identificar alguno, *lex* imprima directamente el aminoácido correspondiente al *stdout*. Como el código de *lex* está optimizado y no se tiene que manejar un buffer, el análisis de las secuencias sería más rápido.

Realizar este cambio no conlleva una gran complejidad ya que sólo habría que adicionar estas reglas y en los casos especiales, en donde algún codón se encuentra separado, utilizar el buffer. Como la gran mayoría de elementos que componen a la cadena son codones válidos, la rápidez del parseo puede aumentar considerablemente.