Informe TP 1

Analizador Lexico genomico

Integrantes:

- Federico Elli
- German Romarion
- Gabriel Zanzotti

Indice:

Introduccion:

El proyecto consiste en crear una gramática genómica que genere un lenguaje que permita expresar una secuencia de nucleótidos que componen el código genético de alguna especie y luego implementar un analizador léxico utilizando LEX para validar las secuencias de nucleótidos.

1. Gramatica

Para obtener la gramática que genera la distintas cadenas de ADN, forzamos a la misma a comenzar con aug, y a terminar con taa, tag o tga. En el medio puede ir cualquier combinación válida de nucleótidos.

Luego para las llaves y corchetes, creamos transiciones acordes a las reglas necesarias. (obs no permitimos cadenas inválidas)

<{S, X}, {a,c,t,g,u}, S, P>
P dado por:
{
S->augT
T->aaaX | aacX | aatX | aagX | acaX | accX | actX | acgX | ataX | atcX | attX | atgX | agaX | agcX | agtX | aggX | caaX | cacX | catX | cagX | ccaX | cccX | cctX | ccgX | ctaX | ctcX | cttX | ctgX | cgaX | cgcX | cgtX | cggX | tacX | tatX | tcaX | tccX | tctX | tcgX | ttaX | ttcX | tttX | ttgX | tgcX | tgtX | tggX | gacX | gatX | gagX | gcaX | gccX | gctX | gcgX | gtaX | gtcX | gttX | gtgX | ggaX | ggcX | ggtX | gggX | taa | tag | tga | stop
X->aaaX | aacX | aatX | aagX | acaX | accX | actX | acgX | ataX | atcX | attX | atgX | agaX | agcX | agtX | aggX | caaX | cacX | catX | cagX | ccaX | cccX | cctX | ccgX | ctaX | ctcX | cttX | ctgX | cgaX | cgcX | cgtX | cggX | tacX | tatX | tcaX | tccX | tctX | tcgX | ttaX | ttcX | tttX | ttgX | tgcX | tgtX | tggX | gacX | gatX | gagX | gcaX | gccX | gctX | gcgX | gtaX | gtcX | gttX | gtgX | ggaX | ggcX | ggtX | gggX | taa | tag | tga | stop
T -> {-}*X
X-> {-}*X
X -> [N]T
N-> N0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | NN
T -> {U}*X
T -> {U}X
U-> a | c | t | g | U,U | UU | {U}
}

Veamos las transiciones necesarias para generar el ejemplo 1 utilizando la gramática propuesta

AUG{G}*[234]{A,G,U}GA{-}*{TTTA,TTTTA,TTT}STOP

S
augT
aug{U}*X
aug{g}*X
aug{g}[N]T
aug{g}[NN]T
aug{g}[NNN]T
aug{g}[2NN]T
aug{g}[23N]T
aug{g}[234]T
aug{g}[234]{U}X
aug{g}[234]{U,U}X
aug{g}[234]{U,U,U}X
aug{g}[234]{a,U,U}X
aug{g}[234]{a,g,U}X
aug{g}[234]{a,g,t}X
aug{g}[234]{a,g,t}X
aug{g}[234]{a,g,t}{-}*X
aug{g}[234]{a,g,t}{-}*{U}X
aug{g}[234]{a,g,t}{-}*{U,U}X
aug{g}[234]{a,g,t}{-}*{U,U,U}X
...
aug{g}[234]{a,g,t}{-}*{UUUU,UUUUUU,UUU}X
...
aug{g}[234]{a,g,t}{-}*{ttta,tttta,ttt}X
aug{g}[234]{a,g,t}{-}*{ttta,tttta,ttt}stop

2. Implementacion con Lex

La implementación con Lex fue planteada desde el punto de vista de la eficiencia y rapidez que ofrece como analizador sintáctico, tratando de aprovechar el mismo al máximo para facilitar el parseo de los distintos codones.

Para lograr este objetivo decidimos generar expresiones regulares diferentes para los codones cuyas dos primeras letras sean diferentes, de esta manera la expresión regular se vuelve mucho más específica y facil de validar. Decidimos hacerlo con las 2 primeras letras de cada nucleótido ya que es la forma más sintética de representar los patrones que LEX luego va a reconocer.

Validamos también que la cadena comience con *AUG* y que termine con alguna de las 4 ternas que generan el codón de stop. Con respecto al resto de las validaciones decidimos tomar que todas las sub cadenas encerradas entre llaves ({ }) y que tuviesen un asterisco (*) sean ignoradas. Optamos por esta solución siendo la misma la más rápida en comparación con las otras soluciones que consideramos (elegir uno específico o elegir al azar).

Con respecto a la expresión con llaves que contiene distintas posibilidades de entrada decidimos usar la solución más simple, tomar siempre la primera opción, descartando el resto de las opciones.

Para validar la posición entre corchetes llevamos un contador que luego se valida contra la posición encontrada.

3. Problemas a la hora de implementar

Nuestra primera implementación consistió en un gran switch/case para cada tipo de combinación de nucleótido. Esta implementación funcionaba para cadenas sin compresión, ni expresiones regulares. La misma era eficiente ya que simplemente era un conversor de codones a Aminoácidos.

Por los problemas mencionados decidimos utilizar los beneficios de análisis sintáctico que brinda LEX. Para ello consideramos todos los posibles casos detallados en el enunciado y construimos un analizador que codifica de forma correcta no solo las cadenas enteras, sino que además acepta distintas elecciones de codones (entre llaves) e identifica cadenas inválidas.

4. Ejemplos adicionales

Ejemplo 6:

Llaves contiguas.

AUGAAA{A,C,T,G}{A,C,T,G}{A,C,T,G}AAATGA

Ejemplo 7:

Spiderman.

AUGTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCT
AATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTC
TCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTA
TTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGAT
GAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAAC
GTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATG
GCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAA
TTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTC
CTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATT
GATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGA
ACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTA
TGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCT
AATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTC
TCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTA
TTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGAT
GAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAAC
GTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATG
GCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAA
TTCTCCTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTC
CTATTGATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTCTCCTATT
GATGAACGTATGGCTAATTCTCCTATTGATGAACGTATGGCTAATTGA

Ejemplo 8:

El siguiente script de bash genera un archivo de aproximadamente 1.1GB para testear el uso de memoria de la aplicación.

```
echo -n "AUG" >> megadna
for i in {1..1000000}
do
    echo -n "ATGACTATTACCATGTTTACACCCATAGAGTTGCCACCAACCTATTACATAAGCCTC
CACATGAATGTGCCGATCTCACCCCTCTATCCATGGGGAATGCCGTATACCCGGTCT
```

ACATCCTTCTCGCATACTACCACCATCTATACGAATACCAATACAAATACCACAACT
ATTGAACGAAAGGATTCGGGAATCTGCATACCATCAAATACCACGTCCCGAAAACC
AAGCACAATTACGATTAATCCCAAGCCAACCACAACTACAACGTACGCGGCGCCTA
GCCGGTGGACAACCCTTACGTCTCCTGCGGAGCAAACAAAGGCTCTTGACTTTCTTA
CTAACAATTATCATACGATTCAGACCTTACAGGCAGCGAACTATACCATTACTCCGG
ATACAACTAAGAATAATATTCGGTTTCAGCAGACGCCAAGCACTAATGACCCTTCTG
CGCGGCGTGCAGTCAGTCTGGACTGCGAAATGGTAACGGATACAACTGGCCGGACT
CAATTGTGCCAAGTAAGTATGGTTGATGTTCTGACAGGCGAACTGCTGCTGGACCAG
CCAGTTCTTCCGAGCGAACCCGTCTCAGACTGGAGAACCAAATGGAGTGGGATGAC
CGCCGAGTTAATGGCCCAGCACGTCGCGGCCGGCCGTACCGTCAACGGGTACGAAG
GAGCGAGAGCACGCGTGTGGGAATACATTGATCAAAAAACAATCTTAGTAGGGCA
GAGTCTCAACTTTGATCTTGACGTTTTAGGCATAGTGCACGAGCGGGTCGTAGACACT
TACCTGCTCATGCGGGGACAGAGACGGGGACGTTCTTGTAGGTTAAGGGATGTCGTC
CGGGACTGCTGCGGAGTCGAGATACAGAAAGGCGAAGAACTCGAGGGGGGGCAT
GATTGTGCGGAAGACTGTTATGCTGCGAGGGAAGTAGCGCTTTGGGCAGTCGAACAT
CTTGGACGAGATGAGGCGTTCATCTCCAGGAGTATGTCCCCCATAGATGACAAATTT
CAGTTGTCGTCTGCATACTTCCCCGCGACCTGTTATTGTATCCCTCGGGATGCTGAGTG
GGCTACTTTTCATGGTTGGACATTGAGGGTGGATACATACGACACCACTCACGTTAAT
TAC" >> megadna
done
echo -n "TAG" >> megadna

Ejemplo 9:

Cadena invalida.

AUG{C,CT,AGA}T{AT,C,G}TGA

Ejemplo10:

Todos los codones

AUGAAAAACAATAAGACAACCACTACGATAATCATTATGAGAAGCAGTAG
GCAACACCATCAGCCACCCCCTCCGCTACTCCTTCTGCGACGCCGTCGGTACT
ATTCATCCTCTTCGTTATTCTTTTTGTGCTGTTGGGAAGACGATGAGGCAGCCG
CTGCGGTAGTCGTTGTGGGAGGCGGTGGGTGA

5. Conclusiones

El analizador léxico *LEX* nos pareció muy útil para identificar patrones y expresiones regulares dentro de una cadena de longitud arbitraria. Sin embargo existen limitaciones cuya solución obliga a la escritura de un programa en C, la complejidad del mismo será proporcional a la magnitud del problema que deba resolverse. Pero a pesar de los problemas y las limitaciones pudimos crear un analizador léxico genómico eficiente (O(n))