MANUAL TECNICO

1. CARACTERES RESERVADOS:

Los caracteres reservados de la expresión regular son:

* | Representa una unión (OR).
* . Indica concatenación (AND); en el programa se puede incluir y/u omitir si se desea.
* \* Indica clausura de kleene.
* + Indica clausura de kleene plus.
* ( Indica e inicio de la agrupación de una secuencia.
* ) Indica el fin de la agrupación de una secuencia.
* \ Indica fin de secuencia; en el usuario se debe incluir el carácter de fin de secuencia al final de la expresión regular.
* @ Indica secuencia vacía.
* $ Indica secuencia nula.

1. PROCESAMIENTO DE EXPRESION REGULAR:

Cuando el usuario escribe una expresión regular en el programa e indica que hay que validarla, el programa almacena cada uno de los caracteres en una estructura genérica llamada “RegexItem”, la cual contiene varios atributos, pero por el momento solo nos interesa que tiene los atributos:

* Primer hijo.
* último hijo.
* Padre.
* Siguiente hermano.
* Anterior hermano.
* Tipo de RegexItem.
* Valor.
* Tipo de kleene (ninguno, kleene “\*”, o kleene plus “+”).

Los tipos de RegexItem que hay son:

1. símbolo (RegexSymbol):

Se refiere a cualquier carácter de entrada que no sea los caracteres reservados, por ejemplo: a, b, c, d, e, f, 0, 1, 2, 3.

Gráficamente un símbolo se vería así, simplemente un nodo.

1. Raíz (RegexRoot):

Es un tipo especial de RegexItem que como su nombre lo indica es la raíz de la expresión regular.

Gráficamente la raíz se vería así, simplemente un nodo sin información.

Si por ejemplo se tiene la expresión regular AB sería así:

Significa que se puede ir de A hacia B y de B hacia A ya que son hermanos, y se pude ir de la raíz a sus hijos y viceversa.

1. Paréntesis (RegexParenthesis):

Un paréntesis se usa para representar una expresión que se encuentra entre paréntesis; dado que un paréntesis tiene una expresión dentro, se usa este RegexItem como si fuese el padre de dicha expresión.

La expresión “(AB)” se representa así:

Si observamos únicamente el nivel 1 (la raíz) no sabemos nada de la expresión regular.

Si observamos el nivel 2 sabremos que la expresión regular en general se compone de un paréntesis “()” y nada más.

Si observamos el nivel 3 ahora si vemos en detalle que hay dentro del paréntesis.

La expresión A(BC) se representa así:

Si analizamos la representación de “A(BC)” vemos que:

Si observamos únicamente el nivel 1 (la raíz) no sabemos nada de la expresión regular.

Si observamos el nivel 2 sabemos que la expresión regular en general se compone de un símbolo seguido de un paréntesis “A()”, pero no sabemos que hay en el paréntesis, entonces “A” es hermano de “()” y ambos son hijos de la raíz. Analizando el nivel 2 se puede ver la forma más general de cualquier expresión regular.

Si observamos el nivel 3 ahora si vemos en detalle que hay dentro del paréntesis, y entonces se ve que “B” y “C” son hermanos.

1. Grupo (RegexGroup):

Es un tipo de RegexItem que no sirve por sí solo, sino en combinación con unión o concatenación, y funciona básicamente igual que un paréntesis, por ejemplo, la expresión “AB” agrupada se vería así:

“<>” es el nodo que representa un grupo genérico, además este mismo RegexItem se podría representar como “{}” si es un grupo de concatenación o “[]” si es un grupo de unión. Esto se sabe observando el atributo Tipo de RegexItem, que todos los RegexItem tienen, pero hasta el momento no nos había sido de utilidad. Los tipos de RegexItem para grupos son:

* + Group: Es el grupo genérico. “<>”
  + GroupU: Es el grupo de unión “[]”.
  + GroupC: Es el grupo de concatenación “{}”.

Más adelante se verán los usos de los grupos.

1. Unión (RegexUnion):

Es un RegexItem auxiliar ya que se usa para representar una unión, pero luego es necesario hacer una conversión a grupos para representar la expresión regular.

Por ejemplo, “A|B|C” se representa así:

Esta representación no funciona siempre ya que si, por ejemplo, tenemos la expresión regular “A|BD|C”:

La anterior representación no es correcta, pues esto representa es “A|B|D|C”, entonces aquí es donde entran los grupos genéricos y los de unión; entonces, una representación correcta sería así:

Para llegar a esta representación se añadieron grupos genéricos a cada uno de los elementos de la unión y se transformó RegexUnion en un grupo de unión, con el fin de que al momento que el programa este haciendo estas transformaciones, sepa que los grupos de unión no es necesario volver a procesarlos.

1. Concatenación (RegexConcat):

Hasta el momento se ha omitido el punto en las expresiones regulares, pero al principio se dijo que existe. Así entonces la expresión “A\*.B+” se vería así:

Igual que con RegexUnion está representación no siempre funciona, pues si por ejemplo, se tiene la expresión “A.BC.D”:

La anterior representación no es exactamente correcta, pues esta representa es “A.B.C.D”, y aunque al final al momento de generar un autómata finito serán equivalentes, para el programa por el momento se podría decir que no lo son.

Para llegar a esta representación se hace lo mismo que con la unión; se añadieron grupos genéricos a cada uno de los elementos de la concatenación y se transformó RegexConcat en un grupo de concatenación, con el fin de que al momento que el programa este haciendo estas transformaciones, sepa que los grupos de concatenación no es necesario volver a procesarlos.

En resumen, después de procesar la cadena con la expresión regular, y validarla, se procede a construir los RegexItem que representan dicha expresión, después se añaden los grupos de unión los grupos de concatenación y los genéricos.

1. CONVERTIR REGEX ITEM EN CONSTRUCCION DE THOMPSON:

Al principio se dijo que cada RegexItem tenía varios atributos, pero que en ese momento no nos interesaban todos, pero ahora sí, y otros atributos son:

* Thompson: Es la construcción de Thompson asociada a dicho RegexItem.
* Símbolo de entrada: Una construcción de Thompson tiene varios estados y varios símbolos, este atributo indica en la construcción de Thompson exactamente a cuál de los símbolos apunta este RegexItem.

Una vez se tienen los RegexItem, se procede a asignar a cada RegexItem la construcción de Thompson a la cual equivale, y el símbolo al que apunta en dicha construcción.

Las construcciones de Thomson se componen de:

1. InputSymbol: Es un símbolo de una construcción de Thompson.
2. State: Es un estado de la construcción de Thompson al cual se llega usando un símbolo.

Para simplificar la creación de las construcciones de Thompson se crearon cada una de las construcciones de Thompson como objetos, y se asignan a cada RegexItem así:

1. Raíz (RegexRoot):

Representación como RegexItem:

Este RegexItem no tiene construcción de Thompson asociada ni símbolo en la construcción.

1. Símbolo (RegexSymbol):

Si se tiene el símbolo A, entonces:

Representación como RegexItem:

Construcción de Thompson asociada:

A

Si el RegexItem es B:

B

Si el RegexItem es lambda:

λ

El nombre de esta construcción de Thompson es ThompsonSymbol, y para el caso de lambda que es una construcción especial se le llama ThompsonNullSymbol.

El símbolo en la construcción Thompson para el caso uno es A, para el dos es B y para el caso tres es λ.

1. Paréntesis (RegexParenthesis):

Funciona exactamente igual que un RegexSymbol.

Representación como RegexItem:

Construcción de Thompson asociada:

()

El nombre de esta construcción de Thompson es ThompsonSymbol

El símbolo de entrada en la construcción de Thompson es ().

Tanto los RegexParenthesis como los RegexSymbol son los únicos RegexItem que luego del procesamiento de la expresión regular pueden tener alguna clausura de kleene.

Por ejemplo, se tiene A+.

Representación como RegexItem:

Construcción de Thompson asociada:

λ

λ A λ

Y si, por ejemplo, se tiene A\*.

Representación como RegexItem:

Construcción de Thompson asociada:

λ

λ A λ

λ

Y el símbolo de entrada en ambas construcciones de Thompson es el objeto tipo InputSymbol donde se encuentra A.

1. Grupo genérico (RegexGroup):

Si se tiene un grupo así:

No se tiene una construcción de Thompson asociada a los grupos genéricos por si solos, estos deben ir en combinación con grupos de unión y o de concatenación.

1. Grupo de unión (RegexGroup):

Se tiene la expresión “A|BD|C”:

La representación como RegexItem es:

La construcción de Thompson asociada es:

<>

λ λ

λ <> λ λ

λ λ

<>

Observando la representación de los RegexItem tenemos que hay un grupo de unión, 3 grupos genéricos y cuatro simbolos, en este caso en la construcción de Thompson vemos que se representan los grupos genéricos, entonces a cada grupo genérico se le asigna la misma construcción de Thompson asociada, pero el símbolo dentro de la construcción para cada grupo es diferente, y para el grupo de unión también se le asigna la misma construcción de Thompson, pero el grupo de unión no tiene un símbolo dentro de la construcción.

Y cada símbolo tiene una construcción de Thompson asociada, y un símbolo dentro de esta como se definió en el inciso b), que recordemos es así:

A

Entonces ahora tenemos que analizando la representación gráfica de los RegexItem vemos que, el nivel uno (la raíz) no tiene construcción de Thompson asociada, el nivel dos y el tres comparten una misma construcción de Thompson, pero el símbolo dentro de la construcción es diferente para cada RegexItem, y en el nivel cuatro tenemos cuatro símbolos, cada uno con su propia construcción de Thompson asociada.

1. Grupo de concatenación (RegexGroup):

Si tenemos la expresión “A.BC.D”, la representación gráfica de los RegexItem es:

Y la construcción de Thompson asociada es:

<> λ <> λ <>

Entonces de manera análoga a como se procedió en el grupo de unión, en el grupo de concatenación también se observa que hay un grupo de concatenación, tres grupos genéricos y cuatro símbolos, y la construcción de Thompson asociada es la misma para los cuatro grupos, pero para el grupo de concatenación no hay símbolo dentro de la construcción de Thompson, y para cada grupo genérico el símbolo dentro de la construcción de Thompson es diferente, y si observamos el nivel cuatro de la representación gráfica de los RegexItem, vemos cuatro símbolos y a cada uno le corresponde una construcción de Thompson asociada, que recordemos se definió en el inciso b) y es así:

A

Llegados a este punto, si se tiene una expresión regular cualquiera, esta se representa como RegexItem, se añaden los grupos de unión y de concatenación, y en este proceso se añaden los genéricos, luego a cada RegexItem se le asigna una construcción de Thompson, y un símbolo dentro de esta, excepto a la raíz, y ahora hay que unir las construcciones de Thompson en una sola.

Si por ejemplo se tiene la expresión “A(BC)+”.

La representación gráfica de los RegexItem es:

Las construcciones de Thompson asociadas al nivel dos son:

A

λ

λ () λ

Las construcciones de Thompson asociadas al nivel tres son:

B

C

Para comenzar con la unión se procede desde el nivel más bajo, en este caso el tres, y se unen las construcciones que sean hermanas.

El proceso de unión consiste en combinar el último estado de un hermano, con el primer estado del siguiente hermano, en este caso luego de la unión la construcción de Thompson del nivel tres sería:

B C

Se unieron el estado S8 y el estado S9 en el estado S89.

Ahora, ya el RegexItem de tipo paréntesis tiene únicamente un hijo, entonces se procede a combinar el padre con su único hijo, para esto se revisa el paréntesis a que símbolo dentro de la construcción de Thompson apunta, que en este caso es el símbolo que está entre los estados S4 y S5.

La unión de un padre con su único hijo consiste en combinar el primer estado del hijo con el estado anterior a donde se encuentra el símbolo del padre dentro de la construcción de Thompson, que en este caso sería unir los estados S4 y S7, y además unir el último estado del hijo con el estado siguiente a donde se encuentra el símbolo del padre dentro de la construcción de Thompson, que para este caso sería unir los estados S5 y S0, dando como resultado:

λ

λ B C λ

La anterior construcción de Thompson se le asigna al RegexItem “()+”, y el símbolo dentro de la construcción no se específica, pues como se ha visto hasta ahora, el símbolo dentro de la construcción básicamente solo se usa para unir un padre con su único hijo una vez que los hijos se unan entre sí.

Ahora ya se eliminó el nivel tres y solo queda el nivel dos con dos hermanos “A” y “()+” y se procede entonces a unirlos en una única construcción combinando el estado S2 y el estado S3 ya que los RegexItem son hermanos.

A λ

λ B C λ

Una vez unidos los estados se procede a unir el padre con su único hijo, pero como el padre es la raíz, se procede únicamente a eliminar el nivel dos y a asignar esta construcción de Thompson a la raíz, terminando el proceso de unión de construcciones de Thompson.

De manera análoga se unen las construcciones de Thompson de cualquier expresión regular representada como RegexItem.

1. CONVERTIR CONSTRUCCION DE THOMPSON EN AUTOMATA FINITO:

Una vez obtenida la construcción de Thompson asociada a una expresión regular el proceso a seguir consiste en obtener el cierra lambda de cada estado.

Retomando la expresión “A(BC)+” con la construcción de Thompson:

A λ

λ B C λ

El cierre lambda de cada estado es:

S1λ = {S1}

S2λ = {S2, S3}

S3λ = {S3}

S4λ = {S4}

S5λ = {S5, S3, S6}

S6λ = {S6}

Transiciones de cada estado con cada símbolo:

S1 ---> A = {S2λ} S1 ---> B = {} S1 ---> C = {}

S2 ---> A = {} S2 ---> B = {} S2 ---> C = {}

S3 ---> A = {} S3 ---> B = {S4λ} S3 ---> C = {}

S4 ---> A = {} S4 ---> B = {} S4 ---> C = {S5λ}

S5 ---> A = {} S5 ---> B = {} S5 ---> C = {}

S6 ---> A = {} S6 ---> B = {} S6 ---> C = {}

Transiciones de cada cierre lambda con cada símbolo:

{S1} ---> A = {S2, S3} rechazo

{S2, S3} ---> B = {S4} rechazo

{S4} ---> C = {S5, S3, S6} rechazo

{S3, S5, S6} ---> B = {S4} aceptación

Ahora renombrando los estados se tiene:

S1 ---> A = S2 rechazo

S2 ---> B = S3 rechazo

S3 ---> C = S4 rechazo

S4 ---> B = S3 aceptación

Las demás transiciones son hacia estado de error.

Ahora lo queda es crear una matriz con cinco filas (cantidad de estados más uno) y cinco columnas (cantidad de símbolos más dos), y se procede a mostrar la matriz en pantalla, quedando así:

Tabla

Descripción generada automáticamente

Donde las celdas en azul indican los estados y los símbolos de entrada, las celdas rojas las transiciones hacia un estado de error, y las celdas blancas las transiciones entre estados y si los estados son o no de aceptación.

Adicionalmente, antes de mostrar el autómata en pantalla se realiza el procedimiento de las particiones para simplificarlo, lo cual en este caso deja el autómata exactamente igual ya que este ya es mínimo.

1. AUTOMATA FINITO:

Una vez generado el autómata, para reconocer una hilera se da clic en archivo y luego en reconocer hilera, lo que abre una caja de texto para ingresar la secuencia a validar si se reconoce. El proceso ya es más sencillo, para reconocer una hilera solo es empezar a ir de un estado a otro, y validar si al terminar de procesar la hilera se encuentra en un estado de aceptación, y seguido mostrar un mensaje en pantalla que indica si se reconoció correctamente la secuencia o no.

1. TECNOLOGÍA:

El programa fue realizado usando el lenguaje de programación JAVA 8 con ayuda del IDE Apache NetBeans y la biblioteca Lombok como proyecto Gradle.

Para ejecutar el programa se abre una consola en la carpeta del proyecto y se usa el comando: “gradlew bootrun” el cual buscará el JAVA\_HOME y verificará que sea 1.8, comenzará con la descarga de Gradle 6.4 en caso de no estar en el sistema y ejecutará el programa; O simplemente se usa el archivo ejecutable de Windows incluido en el proyecto.