UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN,

CAMPUS I

NOMBRE DEL ALUMNO (A):

Deycy Mercedes López Peñate.

SEMESTRE: 6 **GRUPO:** “M”

MATRICULA: A210524

LICENCIATURA:

Ingeniería en desarrollo y tecnologías de software.

NOMBRE DEL DOCENTE:

Dr. Luis Gutiérrez Alfaro.

MATERIA:

Compiladores.

ACTIVIDAD:

Actividad 1. Investigación y ejemplos.

LUGAR:

Tuxtla Gutiérrez, Chiapas.

FECHA DE ENTREGA:

27 / 01 / 2023

CONTENIDO

CONCEPTO DE EXPRESION REGULAR 3
I.- EXPLICAR LOS TIPOS DE OPERADORES DE EXPRESIONES REGULARES. 5
II.- EXPLICAR EL PROCESO DE CONVERSIÓN DE DFA A EXPRESIONES REGULARES. 5
III.- EXPLICAR LEYES ALGEBRAICAS DE EXPRESIONES REGULARES. 6
BIBLIOGRAFIA 7

CONCEPTO DE EXPRESION REGULAR

Las expresiones regulares son un equivalente algebraico para un autómata. Utilizado en muchos lugares como un lenguaje para describir patrones en texto que son sencillos pero muy útiles. Pueden definir exactamente los mismos lenguajes que los autómatas pueden describir: Lenguajes regulares. Además, ofrecen algo que los autómatas no: Manera declarativa de expresar las cadenas que queremos aceptar.

Sirven como lenguaje de entrada a muchos sistemas que procesan cadenas tales como:

- Comandos de búsqueda, e.g., grep de UNIX
- Sistemas de formateo de texto: Usan notación de tipo expresión regular para describir patrones
- Convierte la expresión regular a un DFA o un NFA y simula el autómata en el archivo de búsqueda
- Generadores de analizadores-léxicos. Como Lex o Flex.
- Los analizadores léxicos son parte de un compilador. Dividen el programa fuente en unidades lógicas (tokens). Tokens como while, números, signos (+, -, <, etc.).
- Produce un DFA que reconoce el token.

De forma más precisa, Las expresiones regulares denotan lenguajes. Por ejemplo, la expresión regular: 01 +10 denota todas las cadenas que son o un 0 seguido de cualquier cantidad de 1s o un 1 seguida de cualquier cantidad de 0s.

La definición de otro autor es que las expresiones regulares En Pascal un identificador está formado por una letra seguida de cero o más letras o dígitos. Ahora mostraremos una notación llamada expresiones regulares que nos permite definir lo anterior con precisión. Utilizaremos la notación tal y como se utiliza en el programa LEX. En ella un identificador de Pascal se describe así:

{letra}({letra}|{dígito}) *

La barra vertical significa OR. Los paréntesis se utilizan para agrupar expresiones. El asterisco significa 0 o más instancias de la expresión entre paréntesis. Las llaves las utilizaremos con dos significados diferentes: para indicar el conjunto compuesto por los strings del interior, y otras veces para indicar que lo que hay dentro no hay que tomarlo textualmente. El significado que utilicemos estará claro por el contexto. Así {letra} indica un carácter alfabético y no la palabra “letra”. La yuxtaposición indica concatenación. Según todo esto la expresión regular anterior puede leerse: Una letra seguida de 0 o más instancias de letras o dígitos.

(Cada expresión regular) se construye de otras expresiones más simples utilizando un conjunto definido de reglas. Cada expresión regular r denota un lenguaje L(r). Las reglas indican la manera de conseguir el conjunto L(r) combinando de varias formas los lenguajes denotados por las subexpresiones de r. Así definimos:

1. ε es una expresión regular para indicar el conjunto formado por la cadena vacía, {ε}.

2. Si a fuese un símbolo del alfabeto, entonces a es una expresión regular para indicar el conjunto formado por la cadena “a”. Es decir {“a”}. De esta manera utilizamos el mismo

símbolo a con tres significados diferentes: Un símbolo de un alfabeto (a), una expresión regular (a), y el conjunto formado por la cadena “a”. Los tres tienen un significado diferente y, en cada momento, quedará claro por el contexto a cuál nos referimos.

3. Supongamos que r y s son dos expresiones regulares que denotan los lenguajes L(r) y L(s). Entonces:

- (r) |(s) es una expresión regular para denotar L(r) ∪ L(s).
- (r)(s) es una expresión regular para denotar L(r)L(s).
- (r)* es una expresión regular que denota a (L(r)) *.
- (r) + es una expresión regular para denotar (L(r)) +.
- (r)? es una expresión para denotar {ε} ∪ L(r). Es decir, cero o 1 ocurrencias de L(r).
- (r) es una expresión regular para indicar L(r). Es decir, podemos colocar paréntesis extras alrededor de cualquier expresión regular que queramos.

A los lenguajes descritos por las expresiones regulares se les denomina conjuntos regulares o lenguajes regulares. Las reglas anteriores son un ejemplo de definición recursiva. Las reglas primera y segunda forman la base de la definición (el caso base), mientras que las reglas del punto tercero constituyen el paso recursivo.

Podemos evitar paréntesis innecesarios si adoptamos el convenio de que:

- Los operadores unarios *, + y? tienen la máxima prioridad.
- La concatenación es la segunda operación en prioridad y es asociativa por la izquierda: ABC = (ab)c.
- La operación | tiene la prioridad mínima y también es asociativa por la izquierda: a|b|c = (a|b)|c.
- Con estos convenios la expresión regular (a)|((b) * (c)) es equivalente a la expresión: a|b * c. Ambas expresiones indican el conjunto de los strings formados por una única a o por cero o más b seguidas de una c. Por ejemplo: Supongamos el alfabeto Σ = {a, b}.
- La expresión regular a|b indica el conjunto {“a”, “b”}.
- La expresión regular (a|b)(a|b) denota el conjunto {“aa ”, “ab”, “ba”, “bb”}
- La expresión regular a* denota el conjunto de todos los strings formados por 0 o más a’s: {ε, “a”, “aa”, “aaa”, . . .}.
- La expresión (a|b) * denota el conjunto formado por todos los strings constituidos por cero o más instancias de a o de b. Es decir, el conjunto de todos los strings formados con las letras a y b. Otra forma de describir este conjunto es como (a *b*) *.
- La expresión ala * b señala el conjunto formado por el String a y todos los strings formados por cero o más ocurrencias de aes y acabados con una b. Puede ocurrir que dos expresiones regulares, r y s, denoten el mismo lenguaje, decimos entonces que ambas expresiones son equivalentes, y escribimos r = s. Por ejemplo, (a|b) = (b|a).

I.- EXPLICAR LOS TIPOS DE OPERADORES DE EXPRESIONES REGULARES.

Comúnmente existen tres operadores de las expresiones regulares las cuales se describen a continuación:

UNION:

- Si L y M son dos lenguajes, su unión se denota por L M e.g. L = 001,10,111, M = ,001, entonces la unión será L M =,10,001,111.

CONCATENACION:

- La concatenación de lenguajes se denota como LM o LM e.g. L = 001,10,111, M = ,001, entonces la concatenación será LM =001,10,111,001001,10001,111001.

CERRADURA:

- Finalmente, la cerradura (o cerradura de Kleene) de un lenguaje L se denota como L. Representa el conjunto de cadenas que puede que pueden formarse tomando cualquier número de cadenas de L, posiblemente con repeticiones y concatenando todas ellas e.g. sí L = 01, L son todas las cadenas con 0s y 1s. Si L = 011, entonces L son todas las cadenas de 0s y 1s tal que los 1s están en pareja. Para calcular L se debe calcular Li para cada i y tomar la unión de todos estos lenguajes. Li tiene 2i elementos. Aunque cada Li es nito, la unión del número de términos de Li es en general un conjunto innato e.g. = o 0=. Generalizando, para todo i mayor o igual que uno, i es el conjunto vaco, no se puede seleccionar ninguna cadena del conjunto vaco.

EJEMPLOS:

1. Si E y F son expresiones regulares, entonces E + F también lo es denotando la unión de L(E) y L(F). L (E +F) = L(E) L(F).

2. Si E y F son expresiones regulares, entonces EF también lo es denotando la concatenación de L(E) y L(F). L(EF) = L(E)L(F).

3. Si E es una expresión regular, entonces E también lo es y denota la cerradura de L(E). Ósea L(E) =(L(E)) *

II.- EXPLICAR EL PROCESO DE CONVERSIÓN DE DFA A EXPRESIONES REGULARES.

Si se necesita demostrar que, para cada expresión regular, existe un autómata nito que acepta el mismo lenguaje. Se selecciona el autómata más apto NFA. Si, por el contrario, se necesita demostrar que por cada autómata nito, hay una expresión regular de siendo

su lenguaje se selecciona el autómata con mayores restricciones: DFA. Los lenguajes aceptados por DFA, NFA, -NFA, RE son llamados lenguajes regulares.

Para convertir expresiones regulares a-NFA se realizan pruebas por inducción en los diferentes operadores (+, concatenación, *) en la expresión regular. Siempre se construye un autómata de una forma especial. Se mostrará que un NFA con transiciones puede aceptar el lenguaje de una RE. Después, se mostrará que un RE puede describir el lenguaje de un DFA (la misma construcción funciona para un NFA). Los lenguajes aceptados por DFA, NFA, -NFA, RE son llamados lenguajes regulares.

Teorema 1 Si L=L(A) para algún DFA A, entonces existe una expresión regular R tal que L = L(R).

Prueba: Suponiendo que A tiene estados {1, 2, n}, n infinito. Tratemos de construir una colección de RE que describan progresivamente conjuntos de rutas del diagrama de transiciones de A

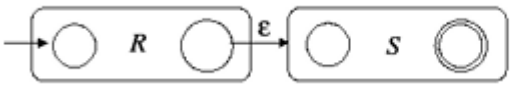
III.- EXPLICAR LEYES ALGEBRAICAS DE EXPRESIONES REGULARES.

Existen un conjunto de leyes algebraicas que se pueden utilizar para las expresiones regulares:

Ley conmutativa para la unión: L + M = M +L Ley asociativa para la unión: (L + M) + N = L+ (M +N)

Ley asociativa para la concatenación: (LM)N = L(MN)

NOTA: La concatenación no es conmutativa, es decir LM= ML



Lenguaje: $L(R)L(S)$

Identidades y aniquiladores

Una identidad para un operador es un valor tal que cuando el operador se aplica a la identidad y a algún otro valor, el resultado es el otro valor. Por ejemplo, 0 es la identidad de la suma, ya que 0+x = x+0 = x, la identidad para la multiplicación es 1 debido a que 1x = x1 = x. Un aniquilador para un operador es un valor tal que cuando el operador se aplica al aniquilador y algún otro valor, el resultado es el aniquilador. 0 es el aniquilador para la multiplicación: 0 x = x 0 = 0. No hay aniquilador para la suma.

Estas leyes las utilizamos para hacer simplificaciones: 0 es la identidad para la union: 0+L = L+0 = L. es la identidad para la concatenación: L = L = L. es el aniquilador para la concatenación: 0L=L0 =0.

Ley distributiva

Como la concatenación no es conmutativa, tenemos dos formas de la ley distributiva para la concatenación:

- Ley Distributiva Izquierda para la concatenación sobre unión: L (M + N) = LM + LN
- Ley Distributiva Derecha para la concatenación sobre unión: (M + N) L = ML +NL

Ley de idempotencia

Se dice que un operador es idempotente (idempotent) si el resultado de aplicarlo a dos argumentos con el mismo valor es el mismo valor. En general la suma no es idempotente: x + x= x (aunque para algunos valores aplica como 0 + 0 = 0). En general la multiplicacion tampoco es idempotente: x *x = x. La unión e intersección son ejemplos comunes de operadores idempotentes. La ley idempotente para la unión: L + L = L.

Leyes relacionadas con la propiedad de cerradura

Las leyes involucradas con la propiedad de cerradura:

- (L*) * =L* (Idempotencia para la cerradura)
- 0* = e
- E*=e
- L+ =LL* =L*, LL+ se define como L+LL+LLL+ …
- L*= L++eL+LL+LLL+…
- LL* =Le+LL+LLL+LLLL+
- L *=L++ e
- L? = +L

Ejemplo:

Probar que (L+M*) = (L*M) Es necesario probar que las cadenas que estan en (L *+ M*) * también estan en (L*M*), y probamos que las cadenas que estan en (LM)* también estan en (L+M) *. Cualquier RE con variables se puede ver como una RE concreta sin variables, viendo cada variable como si fuera un símbolo diferente. La expresión (L + M) se puede ver cómo (a +b) *. Utilizamos esta forma como una gua para concluir sobre los lenguajes. Podemos analizar el lenguaje que nos describe: (a +b) * y analizar el lenguaje que nos describe: (a*b*).

BIBLIOGRAFIA

Jiménez Millán, J. A. (2014). Compiladores y procesadores de lenguajes: (ed.). Cádiz, Spain: Servicio de Publicaciones de la Universidad de Cádiz. Recuperado de https://elibro.net/es/ereader/uachiapas/33847?.

Ciencias computacionales Propedéutico: TeoradeAutomatasyLenguajesFormales Expresionesregularesylenguajes. Inaoep.mx.