Universidad Autónoma de Chiapas

Facultad de Contaduría y Administración, Campus I

Licenciatura en Ingeniera en Desarrollo y Tecnologías de Software

Asignatura: Compiladores

Actividad I. Investigación y Ejemplos.

Dr. Luis Gutiérrez Alfaro

Alumno: Aldrin Aquino Sánchez Matricula: A221681

Semestre: 6º Grupo: M

Tuxtla Gutiérrez, Chiapas.

A 15 de agosto del 2024.

Actividad I.- Investigación y Ejemplos.

Definir los siguientes Conceptos y de ejemplo de cada uno de los Incicios de I, II, III.

Definir el concepto de expresión regular.

Son las unidades de descripción de los lenguajes regulares, que se incluyen en los denominados lenguajes formales. Son un instrumento clave de la informática teórica, la cual, entre otras cosas, establece las bases para el desarrollo y la ejecución de programas informáticos, así como para la construcción del compilador necesario para ello. Es por esto que las expresiones regulares, también denominadas regex y basadas en reglas sintácticas claramente definidas, se utilizan principalmente en el ámbito del desarrollo de software.

I.- Explicar los tipos de operadores de expresiones regulares.

^ Indica el principio de una cadena

$ Indica el final de una cadena

() Un agrupamiento de parte de una expresión

[] Un conjunto de caracteres de la expresión

{} Indica un número o intervalo de longitud de la expresión

. Cualquier carácter salvo el salto de línea

? 0-1 ocurrencias de la expresión

+ 1-n ocurrencias de la expresión

* 0-n ocurrencias de la expresión

\ Para escribir un carácter especial como los anteriores y que sea tratado como un literal

| Para indicar una disyunción lógica (para elegir entre dos valores: a|b se tiene que cumplir al menos uno de los dos)

Ejemplos básicos:

[A-Za-z0-9]+$: Permite caracteres de la A-Z, tanto en minúsculas como mayúsculas y dígitos del 0-9

z{3} : debe de tener 3 concidencias consecutivas de la letra "z"{zzz}

\d: solo números {0,1,2,3,4,5,6,7,8,9}

go*gle: la letra "o" puede no estar una o más veces {ggle, gogle, google, gooogle, goooogle, ...}

go+gle: la letra "o" puede estar una o más veces {gogle, google, gooogle, goooogle, ...}

gray|grey: contiene una de esta dos palabra {gray, grey}

colou?r: la letra u es opcional {color, colour}

Escapes de carácter

El carácter de barra diagonal inversa (\) en una expresión regular indica que el carácter que lo sigue es un carácter especial (como se muestra en la tabla siguiente) o que se debe interpretar literalmente.

Ejemplo:

- \t : Coincide con una tabulación
- \n : Coincide con una nueva línea
- \r : Coincide con un retorno de carro
- \d : Coincide con dígitos

Clases de caracteres

Una clase de caracteres coincide con cualquiera de un juego de caracteres.

- [character_group] Coincide con cualquier carácter individual de character_group. De forma predeterminada, la coincidencia distingue entre mayúsculas y minúsculas.
- [^character_group] Negación: coincide con cualquier carácter único que no esté en character_group. De forma predeterminada, los caracteres de character_group distinguen entre mayúsculas y minúsculas.
- [Primero-Última] Intervalo de caracteres: coincide con cualquier carácter individual del intervalo de primero a último.
- . Carácter comodín: coincide con cualquier carácter único excepto \n.
- \w Coincide con cualquier carácter de palabra.
- \W Coincide con cualquier carácter que no sea de palabra.
- \s Coincide con cualquier carácter de espacio en blanco.
- \S Coincide con cualquier carácter que no sea de espacio en blanco.
- \d Coincide con cualquier dígito decimal.
- \D Coincide con cualquier carácter distinto de un dígito decimal.

Delimitadores

Los delimitadores, hacen que una coincidencia tenga éxito o no dependiendo de la posición actual en la cadena, pero no hacen que el motor avance por la cadena ni consuma caracteres.

- ^ De forma predeterminada, la coincidencia debe comenzar al principio de la cadena; en el modo multilínea, debe comenzar al principio de la línea.
- $ De forma predeterminada, la coincidencia se debe producir al final de la cadena o antes de \n al final de la cadena; en el modo multilínea, se debe producir antes del final de la línea o antes de \n al final de la línea.
- \A La coincidencia se debe producir al principio de la cadena.

- \Z La coincidencia se debe producir al final de la cadena o antes de \n al final de la cadena.
- \z La coincidencia se debe producir al final de la cadena.
- \G La coincidencia debe producirse en el punto en el que finalizó la coincidencia anterior, o si no había ninguna coincidencia anterior, en la posición de la cadena donde se inició la coincidencia.
- \b La coincidencia se debe producir en un límite entre un carácter \w (alfanumérico) y un carácter \W (no alfanumérico).
- \B La coincidencia no se debe producir en un límite \b.

Construcciones de agrupamiento

Las construcciones de agrupamiento definen subexpresiones de una expresión regular y, normalmente, capturan subcadenas de una cadena de entrada.

- (Subexpresión) Captura la subexpresión coincidente y le asigna un número ordinal basado en uno.
- (?=Subexpresión) Aserción de búsqueda anticipada positiva de ancho cero.
- (?<=Subexpresión) Aserción de búsqueda tardía positiva de ancho cero.
- (?<!Subexpresión) Aserción de búsqueda tardía negativa de ancho cero.

Cuantificadores

Un cuantificador especifica cuántas instancias del elemento anterior (que puede ser un carácter, un grupo o una clase de caracteres) debe haber en la cadena de entrada para que se encuentre una coincidencia.

- * Coincide con el elemento anterior cero o más veces.
- + Coincide con el elemento anterior una o más veces.
- ? Coincide con el elemento anterior cero veces o una vez.
- {N} Coincide con el elemento anterior exactamente n veces.
- {N,} Coincide con el elemento anterior al menos n veces.
- {N,M} Coincide con el elemento anterior al menos n veces, pero no más de m veces.

II.- Explicar el proceso de conversión de DFA a expresiones regulares.

Un autómata determinista de estado finito (DFA) es un modelo matemático utilizado para reconocer y describir lenguajes regulares. Consiste en un conjunto finito de estados, un conjunto de símbolos de entrada, una función de transición, un estado inicial y un conjunto de estados de aceptación. Los DFA se utilizan ampliamente en varios campos, incluida la ciberseguridad, ya que proporcionan una forma formal y sistemática de analizar y manipular lenguajes regulares.

Convertir un DFA en una expresión regular equivalente es un concepto fundamental en la teoría de la complejidad computacional. Nos permite expresar el mismo lenguaje reconocido por el DFA usando una notación más concisa y expresiva.

Para convertir un DFA en una expresión regular equivalente, podemos usar el método de eliminación de estados. La idea básica es eliminar sistemáticamente los estados del DFA mientras se preserva el idioma que reconoce. Este método consta de los siguientes pasos:

1. Eliminar todos los estados de no aceptación: comience eliminando todos los estados de no aceptación del DFA. Dado que estos estados no contribuyen al lenguaje reconocido por DFA, podemos eliminarlos de manera segura sin afectar la expresión regular final.

2. Introduzca un nuevo estado de inicio: cree un nuevo estado de inicio y agregue ε-transiciones desde este estado a cada uno de los estados de inicio originales. Esto garantiza que la expresión regular incluya todas las rutas posibles desde el nuevo estado de inicio hasta los estados de aceptación originales.

3. Eliminar estados uno por uno: Para cada estado restante en el DFA, lo eliminamos redirigiendo las transiciones entrantes y salientes a través de un nuevo estado. Este nuevo estado representa la expresión regular que describe las rutas entre los estados que se eliminan.

a. Redirigir las transiciones entrantes: para cada transición entrante al estado que se elimina, cree una nueva transición del estado de origen al estado de destino, etiquetada con la concatenación de la etiqueta de transición original y la expresión regular que representa el estado que se elimina.

b. Redirigir las transiciones salientes: para cada transición saliente del estado que se elimina, cree una nueva transición del estado de origen al estado de destino, etiquetada con la expresión regular que representa el estado que se elimina.

4. Repita el paso 3 hasta que solo queden dos estados: Continúe eliminando estados uno por uno hasta que solo queden dos estados en el DFA. Estos dos estados representan la expresión regular final que describe el lenguaje reconocido por el DFA original.

5. Combine los dos estados finales: combine los dos estados finales en un solo estado, etiquetado con la expresión regular que representa el idioma reconocido por el DFA original. Esta expresión regular es la forma equivalente del DFA original.

Ilustremos este proceso de conversión con un ejemplo:

Considere un DFA con tres estados: A, B y C. El estado A es el estado de inicio, el estado C es el único estado de aceptación y la función de transición es la siguiente:

– A -> B (entrada: 0)

– B -> C (entrada: 1)

– C -> B (entrada: 0, 1)

Para convertir este DFA en una expresión regular equivalente, seguimos los pasos descritos anteriormente:

Paso 1: eliminar el estado de no aceptación A.

Paso 2: Introduzca un nuevo estado inicial S y agregue transiciones ε de S a A.

Paso 3: Elimina el estado B.

– Redirigir las transiciones entrantes: S -> C (entrada: 0)

– Redirigir transiciones salientes: C -> C (entrada: 1)

Paso 4: combine los estados S y C en un solo estado etiquetado con la expresión regular (0(1(0,1)*)).

La expresión regular resultante describe el idioma reconocido por el DFA original.

Convertir un autómata de estado finito determinista (DFA) en una expresión regular equivalente implica eliminar estados sistemáticamente y preservar el lenguaje reconocido por el DFA. Este proceso permite una representación más concisa y expresiva del lenguaje regular. El método de eliminación de estado es un concepto fundamental en la teoría de la complejidad computacional y proporciona un enfoque sistemático para convertir DFA en expresiones regulares.

III.- Explicar leyes algebraicas de expresiones regulares.

1. Cierre:

si r1 y r2 son expresiones regulares (RE), entonces

- r1* es un RE
- r1+r2 es un RE
- r1.r2 es un RE

2. Leyes de cierre –

- (r*)* = r, cerrar una expresión que ya está cerrada no cambia el idioma.
- ∅* = ∈, una string formada al concatenar cualquier número de copias de una string vacía está vacía.
- r + = rr* = r*r, como r* = ∈ + r + rr+ rrr …. y rr* = r+ rr + rrr ……
- r* = r*+ ∈

3. Asociatividad:

si r1, r2, r3 son RE, entonces

i.) r1+ (r2+r3) = (r1+r2) +r3

- Por ejemplo : r1 = a, r2 = b, r3 = c, entonces
- La expresión regular resultante en LHS se convierte en a+(b+ c) y el conjunto regular para el RE correspondiente es {a, b, c}.

- porque el RE en RHS se convierte en (a+ b) + c y el conjunto regular para este RE es {a, b, c}, que es el mismo en ambos casos. Por lo tanto, la propiedad de asociatividad se cumple para el operador de unión.

ii.) r1.(r2.r3) = (r1.r2).r3

- Por ejemplo – r1 = a , r2 = b , r3 = c
- Entonces la string aceptada por RE a.(bc) es solo abc.
- La string aceptada por RE en RHS es (ab).c es solo abc, que es igual en ambos casos. Por lo tanto, la propiedad de asociatividad se cumple para el operador de concatenación.

La propiedad de asociatividad no se cumple para el cierre de Kleene (*) porque es un operador unario.

4. Identidad –

En el caso de operadores de unión

si r+ x = r ⇒ x= ∅ como r ∪ ∅= r, entonces ∅ es la identidad de +.

Por lo tanto, ∅ es el elemento de identidad para un operador de unión.

En el caso del operador de concatenación,

si rx = r, para x= ∈

r.∈ = r ⇒ ∈ es el elemento de identidad para el operador de concatenación (.) .

5. Aniquilador –

- Si r+ x = r ⇒ r ∪ x= x , no hay aniquilador para +
- En el caso de un operador de concatenación, rx = x, cuando x = ∅, entonces r.∅ = ∅, por lo tanto, ∅ es el aniquilador del operador (.). Por ejemplo {a, aa, ab}.{ } = { }

6. Propiedad conmutativa:

si r1, r2 son RE, entonces

- r1+r2 = r2+r1. Por ejemplo, para r1 =a y r2 =b, entonces RE a+ b y b+ a son iguales.
- r1.r2 ≠ r2.r1. Por ejemplo, para r1 = a y r2 = b, entonces RE ab no es igual a ba

7. Propiedad distribuida:

si r1, r2, r3 son expresiones regulares, entonces

- (r1+r2).r3 = r1.r3 + r2.r3 es decir, distribución correcta
- r1.(r2+ r3) = r1.r2 + r1.r3 es decir, distribución izquierda
- (r1.r2) +r3 ≠ (r1+r3)(r2+r3)

8. Ley idempotente –

- r1 + r1 = r1 ⇒ r1 ∪ r1 = r1 , por lo tanto, el operador de unión satisface la propiedad idempotente.
- El operador de concatenación rr ≠ r ⇒ no satisface la propiedad idempotente.

9. Identidades para la expresión regular:

Hay muchas identidades para la expresión regular. Sean p, q y r expresiones regulares.

- ∅ + r = r
- ∅.r= r.∅ = ∅
- ∈.r = r.∈ =r
- ∈* = ∈ y ∅* = ∈
- r + r = r
- r*.r* = r*
- rr* = r*.r = r + .
- (r*)* = r*
- ∈ +rr* = r* = ∈ + rr*
- (pq)*.p = p.(qp)*
- (p + q)* = (p*.q*)* = (p* + q*)*
- (p+ q).r= p.r+ qr y r.(p+q) = rp + rq

Referencias

Equipo editorial de IONOS. (2019, 30 diciembre). *Regex o expresiones regulares: la manera más sencilla de describir secuencias de caracteres*. IONOS Digital Guide. https://www.ionos.mx/digitalguide/paginas-web/creacion-de-paginas-web/regex/

Cabrera, E. (2024, 15 marzo). *RegEx 101: Guía de supervivencia para entender y usar expresiones regulares*. Eudris Cabrera Personal Site. https://eudriscabrera.com/blog/2022/regex-101.html

Academy, E. (2023, 2 agosto). *¿Cómo se puede convertir un autómata determinista de estado finito (DFA) en una expresión regular equivalente? - Academia EITCA*. EITCA Academy. https://es.eitca.org/cybersecurity/eitc-is-cctf-computational-complexity-theory-fundamentals/regular-languages/equivalence-of-regular-expressions-and-regular-languages/examination-review-equivalence-of-regular-expressions-and-regular-languages/how-can-a-deterministic-finite-state-automaton-dfa-be-converted-into-an-equivalent-regular-expression/

Greyrat, R. (2022, 5 julio). *Propiedades de las expresiones regulares – Barcelona Geeks*. https://barcelonageeks.com/propiedades-de-las-expresiones-regulares/