# ACTIVIDAD I

## COMPILADORES

Alumno: Alfredo López Méndez
Asesor: Dr. Luis Gutiérrez Alfaro

**Universidad Autónoma de Chiapas**
Facultad de Contaduría y Administración Campus I
Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software
Tuxtla Gutiérrez, Chiapas

5to Semestre; Grupo M
Entrega:
jueves, 15 de agosto de 2024

**Universidad Autónoma de Chiapas**
Facultad de Contaduría y Administración Campus I
Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software
Tuxtla Gutiérrez, Chiapas

# Actividad I. Investigación y Ejemplos.

## Compiladores

### Define el concepto de expresión regular

"Una expresión regular es un modelo con el que el motor de expresiones regulares intenta buscar una coincidencia en el texto de entrada. Un modelo consta de uno o más literales de carácter, operadores o estructuras" (Microsoft, 2024.).

### Explicar los tipos de operadores de expresiones regulares.

#### Escapes de carácter

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

| Carácter de escape | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `\a` | Coincide con un carácter de campana, \u0007. | `\a` | `"\u0007"` en `"Error!" + '\u0007'` |
| `\b` | En una clase de caracteres, coincide con un retroceso, \u0008. | `[\b]{3,}` | `"\b\b\b\b"` en `"\b\b\b\b"` |
| `\t` | Coincide con una tabulación, \u0009. | `(\w+)\t` | `"item1\t"`, `"item2\t"` en `"item1\titem2\t"` |
| `\r` | Coincide con un retorno de carro, \u000D. (`\r` no es equivalente al carácter de nueva línea, `\n`). | `\r\n(\w+)` | `"\r\nThese"` en `"\r\nThese are\ntwo lines."` |

#### Clases de caracteres

Una clase de caracteres coincide con cualquiera de un juego de caracteres. Son como una caja llena de caracteres diferentes.

- Cuando usas una, es como si dijeras "Dame cualquiera de estos, me vale con uno".
- Muy útiles cuando buscas "alguna vocal" o "cualquier número".

| Clase de carácter | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `[` *grupo_caracteres* `]` | Coincide con cualquier carácter individual de *character_group*. De forma predeterminada, la coincidencia distingue entre mayúsculas y minúsculas. | `[ae]` | `"a"` en `"gray"`<br><br>`"a"`, `"e"` en `"lane"` |
| `[^` *grupo_caracteres* `]` | Negación: coincide con cualquier carácter individual que no esté en *character_group*. De forma predeterminada, los caracteres de *grupo_caracteres* distinguen entre mayúsculas y minúsculas. | `[^aei]` | `"r"`, `"g"`, `"n"` en `"reign"` |
| `[` *primero* `-` *último* `]` | El intervalo de caracteres: coincide con cualquier carácter individual en el intervalo de *primero* a *último*. | `[A-Z]` | `"A"`, `"B"` en `"AB123"` |
| `.` | Carácter comodín: coincide con cualquier carácter individual a excepción de \n.<br><br>Para que coincida un carácter de punto literal (o `\u002E`), debe ir precedido de un carácter de escape (`\.`). | `a.e` | `"ave"` en `"nave"`<br><br>`"ate"` en `"water"` |

## Delimitadores

Los delimitadores, o aserciones atómicas de ancho cero, 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.

| Aserción | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `^` | De forma predeterminada, la coincidencia debe comenzar al principio de la cadena; en el modo multilínea, debe comenzar al principio de la línea. | `^\d{3}` | `"901"` en `"901-333-"` |
| `$` | De forma predeterminada, la coincidencia se debe producir al final de la cadena o antes de `\n` al final de la cadena; en el modo multilínea, se debe producir antes del final de la línea o antes de `\n` al final de la línea. | `-\d{3}$` | `"-333"` en `"-901-333"` |
| `\A` | La coincidencia se debe producir al principio de la cadena. | `\A\d{3}` | `"901"` en `"901-333-"` |
| `\Z` | La coincidencia se debe producir al final de la cadena o antes de `\n` al final de la cadena. | `-\d{3}\Z` | `"-333"` en `"-901-333"` |
| `\z` | La coincidencia se debe producir al final de la cadena. | `-\d{3}\z` | `"-333"` en `"-901-333"` |

## Construcciones de agrupamiento

Las construcciones de agrupamiento definen subexpresiones de una expresión regular y, normalmente, capturan subcadenas de una cadena de entrada. Funcionan como paréntesis en una frase, agrupando partes de la expresión.

| Construcción de agrupamiento | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `(` *subexpresión* `)` | Captura la subexpresión coincidente y le asigna un número ordinal basado en uno. | `(\w)\1` | `"ee"` en `"deep"` |
| `(?<` *nombre* `>` *subexpresión* `)` o bien `(?'` *nombre* `'` *subexpresión* `)` | Captura la subexpresión coincidente en un grupo con nombre. | `(?<double>\w)\k<double>` | `"ee"` en `"deep"` |
| `(?<` *nombre1* `-` *nombre2* `>` *subexpresión* `)` o bien `(?'` *nombre1* `-` *nombre2* `'` *subexpresión* `)` | Define una definición de grupo de equilibrio. Para obtener más información, consulte la sección "Definiciones de grupos de equilibrio" en Construcciones de agrupamiento. | `(((?'Open'\()[^\(\)]*)+ ((?'Close-Open'\))[^\(\)]*)+)* (?(Open)(?!))$` | `"((1-3)*(3-1))"` en `"3+2^((1-3)*(3-1))"` |

## Introducción a búsqueda

Busca pistas específicas antes o después de un punto determinado.

• Puede confirmar si encuentra o no lo que está buscando, sin moverse de su posición.
• El resultado de esta subexpresión se evalúa en función de si es una aserción positiva o negativa.

| Búsqueda | Nombre | Función |
|---|---|---|
| `(?=check)` | Búsqueda posterior positiva | Afirma que lo que va inmediatamente después de la posición actual de la cadena es "check" |
| `(?<=check)` | Búsqueda anterior positiva | Afirma que lo que va inmediatamente antes de la posición actual de la cadena es "check" |
| `(?!check)` | Búsqueda posterior negativa | Afirma que lo que va inmediatamente después de la posición actual de la cadena no es "check" |
| `(?<!check)` | Búsqueda anterior negativa | Afirma que lo que va inmediatamente antes de la posición actual de la cadena no es "check" |

## 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.

| Cuantificador | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `*` | Coincide con el elemento anterior cero o más veces. | `a.*c` | `"abcbc"` en `"abcbc"` |
| `+` | Coincide con el elemento anterior una o más veces. | `"be+"` | `"bee"` en `"been"`, `"be"` en `"bent"` |
| `?` | Coincide con el elemento anterior cero veces o una vez. | `"rai?"` | `"rai"` en `"rain"` |
| `{ n }` | Coincide con el elemento anterior exactamente *n* veces. | `",\d{3}"` | `",043"` en `"1,043.6"`, `",876"`, `",543"` y `",210"` en `"9,876,543,210"` |
| `{ n ,}` | Coincide con el elemento anterior al menos *n* veces. | `"\d{2,}"` | `"166"`, , `"29"`, `"1930"` |
| `{ n , m }` | Coincide con el elemento anterior al menos *n* veces, pero no más de *m* veces. | `"\d{3,5}"` | `"166"`, `"17668"`<br><br>`"19302"` en `"193024"` |

## Construcciones de referencia inversa

Una referencia inversa permite identificar una subexpresión coincidente previamente más adelante en la misma expresión regular. Muy útil para encontrar palabras repetidas o patrones que se repiten.

| Construcción de referencias inversas | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `\` *número* | Referencia inversa Coincide con el valor de una subexpresión numerada. | `(\w)\1` | `"ee"` en `"seek"` |
| `\k<` *nombre* `>` | Referencia inversa con nombre Coincide con el valor de una expresión con nombre. | `(?<char>\w)\k<char>` | `"ee"` en `"seek"` |

## Construcciones de alternancia

Las estructuras de alternancia modifican una expresión regular para habilitar o no la coincidencia. Es como tener un menú de opciones en tu búsqueda.

| Construcciones de alternancia | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `\|` | Coincide con cualquier elemento separado por el carácter de barra vertical (`\|`). | `th(e\|is\|at)` | `"the"`, `"this"` en `"this is the day."` |
| `(?(` *expresión* `)` *sí* `\|` *no* `)`<br>o bien<br>`(?(` *expresión* `)` *Sí* `)` | Coincide con *sí* si el patrón de expresión regular designado por *expresión* coincide; de lo contrario, coincide con la parte opcional *no*. *expresión* se interpreta como una aserción de ancho cero.<br><br>Para evitar ambigüedades con un grupo de captura con nombre o numerado, puede usar una aserción explícita, como la siguiente:<br>`(?(` `(?=` *expresión* `)` `)` *sí* `\|` *no* `)` | `(?(A)A\d{2}\b\|\b\d{3}\b)` | `"A10"`, `"910"` en `"A10 C103 910"` |
| `(?(` *nombre* `)` *sí* `\|` *no* `)`<br>o bien<br>`(?(` *nombre* `)` *Sí* `)` | Coincide con *sí* si *nombre*, un grupo de captura con nombre o numerado, tiene una coincidencia; de lo contrario, coincide con la parte opcional *no*. | `(?<quoted>")?(?(quoted).+?"\|\S+\s)` | `"Dogs.jpg "`, `"\"Yiska playing.jpg\""` en `"Dogs.jpg \"Yiska playing.jpg\""` |

## Sustituciones

Las sustituciones son elementos del lenguaje de expresiones regulares que se admiten en modelos de reemplazo. Son las herramientas que usas cuando quieres cambiar el texto, no solo encontrarlo.

• Te permiten decir "cuando encuentres X, cámbialo por Y".

| Carácter | Descripción | Modelo | Modelo de reemplazo | Cadena de entrada | Cadena de resultado |
|---|---|---|---|---|---|
| `$` *número* | Sustituye la subcadena que coincide con el grupo *número*. | `\b(\w+)(\s)(\w+)\b` | `$3$2$1` | `"one two"` | `"two one"` |
| `${` *nombre* `}` | Sustituye la subcadena que coincide con el grupo con nombre *nombre*. | `\b(?<word1>\w+)(\s)(?<word2>\w+)\b` | `${word2}${word1}` | `"one two"` | `"two one"` |
| `$$` | Sustituye un "$" literal. | `\b(\d+)\s?USD` | `$$$1` | `"103 USD"` | `"$103"` |
| `$&` | Sustituye una copia de toda la coincidencia. | `\$?\d*\.?\d+` | `**$&**` | `"$1.30"` | `"**$1.30**"` |
| `$`` | Sustituye todo el texto de la cadena de entrada delante de la coincidencia. | `B+` | `$`` | `"AABBCC"` | `"AAAACC"` |
| `$'` | Sustituye todo el texto de la cadena de entrada detrás de la coincidencia. | `B+` | `$'` | `"AABBCC"` | `"AACCCC"` |
| `$+` | Sustituye el último grupo capturado. | `B+(C+)` | `$+` | `"AABBCCDD"` | `"AACCDD"` |
| `$_` | Sustituye toda la cadena de entrada. | `B+` | `$_` | `"AABBCC"` | `"AAAABBCCCC"` |

## Opciones de expresiones regulares

Puede especificar opciones que controlan cómo debe interpretar el motor de expresiones regulares un patrón de expresión regular. Son como los ajustes de un buscador de texto.

• Te permiten afinar cómo se interpreta tu expresión regular.

• Pueden hacer cosas como ignorar mayúsculas y minúsculas o cambiar cómo se tratan los saltos de línea.

| Opción | Descripción | Modelo | Coincidencias |
|---|---|---|---|
| `i` | Usa la coincidencia sin distinción entre mayúsculas y minúsculas. | `\b(?i)a(?-i)a\w+\b` | `"aardvark"`, `"aaaAuto"` en `"aardvark AAAuto aaaAuto Adam breakfast"` |
| `m` | Usa el modo multilínea. `^` y `$` coinciden con el principio y el final de una línea, en lugar del principio y el final de una cadena. | Para obtener un ejemplo, consulte la sección "Modo multilínea" en Opciones de expresiones regulares. | |
| `n` | No se capturan grupos sin nombre. | Para obtener un ejemplo, consulte la sección "Solo capturas explícitas" en Opciones de expresiones regulares. | |

## Explicar el proceso de conversión de DFA a expresiones regulares.

Un autómata determinista de estado finito (DFA) es un modelo matemático para reconocer lenguajes regulares. Convertir un DFA a una expresión regular equivalente es importante en la teoría de la complejidad computacional, permitiendo expresar el mismo lenguaje de forma más concisa. El método de eliminación de estados se usa para esta conversión, preservando el lenguaje reconocido mientras se eliminan sistemáticamente los estados del DFA.

1. **Eliminar estados de no aceptación:**

   Primero, quitamos todos los estados que no son de aceptación del DFA. Estos estados no afectan el lenguaje reconocido, por lo que su eliminación no cambia la expresión regular final.

2. **Agregar un nuevo estado de inicio:**

   Creamos un nuevo estado inicial y conecta este estado con los estados iniciales originales mediante transiciones ε (epsilon). Esto asegura que todas las posibles rutas desde el nuevo inicio hasta los estados de aceptación sean consideradas.

3. **Eliminar estados paso a paso:**

   Para cada estado restante, eliminamos y ajustamos las transiciones entrantes y salientes:

   - Redirigir transiciones entrantes: Para cada transición que llega al estado eliminado, creamos una nueva transición desde el estado de origen al de destino, etiquetada con la concatenación de la etiqueta original y la expresión regular del estado eliminado.
   - Redirigir transiciones salientes: Para cada transición que sale del estado eliminado, creamos una nueva transición desde el estado de origen al de destino, etiquetada con la expresión regular del estado eliminado.

4. **Repetir hasta dos estados:**

   Continuamos eliminando estados uno por uno hasta que solo queden dos estados en el DFA. Estos dos estados representan la expresión regular que describe el lenguaje del DFA original.

5. **Combinar los dos estados finales:**

   Combinamos los dos estados restantes en uno solo, etiquetado con la expresión regular que representa el lenguaje reconocido por el DFA original. Esta expresión regular es la versión simplificada del DFA original.

**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: Introducir un nuevo estado inicial S y agregar transiciones ε de S a A.

Paso 3: Eliminar 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)*)).

## Explicar leyes algebraicas de expresiones regulares.

1. **Conmutativas:**

Se dice que un lenguaje L es conmutativo si se cumple que un operador pueda cambiar el orden de sus operadores y aun así obtener el mismo resultado.

*L+M = M +L*. Esta ley, la ley conmutativa de la unión, establece que podemos efectuar la unión de dos lenguajes en cualquier orden.

2. **Asociativas:**

La asociativo es la propiedad de un operador que nos permite reagrupar los operando cuando el operador se aplica dos veces.

*(L+M)+N = L+ (M +N)*. Esta ley, la ley asociativa para la unión, establece que podemos efectuar la unión de tres lenguajes bien calculando primero la unión de los dos primeros, o bien la unión de los dos últimos.

3. **Elemento Identidad:**

El elemento identidad de un operador es un valor que operado con cualquier otro número no lo altera.

Ejemplo: 0 es el elemento identidad para la suma, ya que $0+X = X+0 = X$, Y 1 es el elemento identidad de la multiplicación, puesto que $1 \times X = X \times 1 = X$.

$0+L=L+/0= L$. Esta ley establece que /0 es el elemento identidad para la unión.

### 4. Elemento Nulo Aniquilador:

Es un valor tal que cuando el operador se aplica al propio elemento nulo y a algún otro valor, el resultado es el elemento nulo.

Ejemplo: 0 es el elemento nulo de la multiplicación, ya que $0 \times x = x \times 0 = 0$.

### 5. Leyes distributivas:

Esta implica a dos operadores y establece que un operador puede aplicarse por separado a cada argumento del otro operador. Existe una ley análoga para las expresiones regulares, que tenemos que establecer de dos formas

$L(M + N) = LM + LN$. Ésta es la ley distributiva por la izquierda de la concatenación respecto de la unión.

$(M +N)L = ML+NL$. Ésta es la ley distributiva por la derecha de la concatenación respecto de la unión.

### 6. Ley de idempotencia:

Se dice que un operador es idempotente si el resultado de aplicarlo a dos valores iguales es dicho valor. Los operadores aritméticos habituales no son idempotentes.

$L + L = L$. Ésta es la ley de idempotencia para la unión, que establece que si tomamos la unión de dos expresiones idénticas, podemos reemplazarla por una copia de la de la expresión.

## Referencias.

EITCA. (2023). ¿Cómo se puede convertir un autómata finito determinista (DFA) en una expresión regular equivalente? 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/

Fandom. (s.f.). Álgebra de las expresiones regulares. Autómatas Wiki. https://automatas.fandom.com/es/wiki/Algebra_de_las_expresiones_regulares

Instituto Nacional de Astrofísica, Óptica y Electrónica. (s.f.). Expresiones regulares. https://ccc.inaoep.mx/~emorales/Cursos/Automatas/ExpRegulares.pdf

Microsoft. (2024). Referencia rápida del lenguaje de expresiones regulares. https://learn.microsoft.com/es-es/dotnet/standard/base-types/regular-expression-language-quick-reference