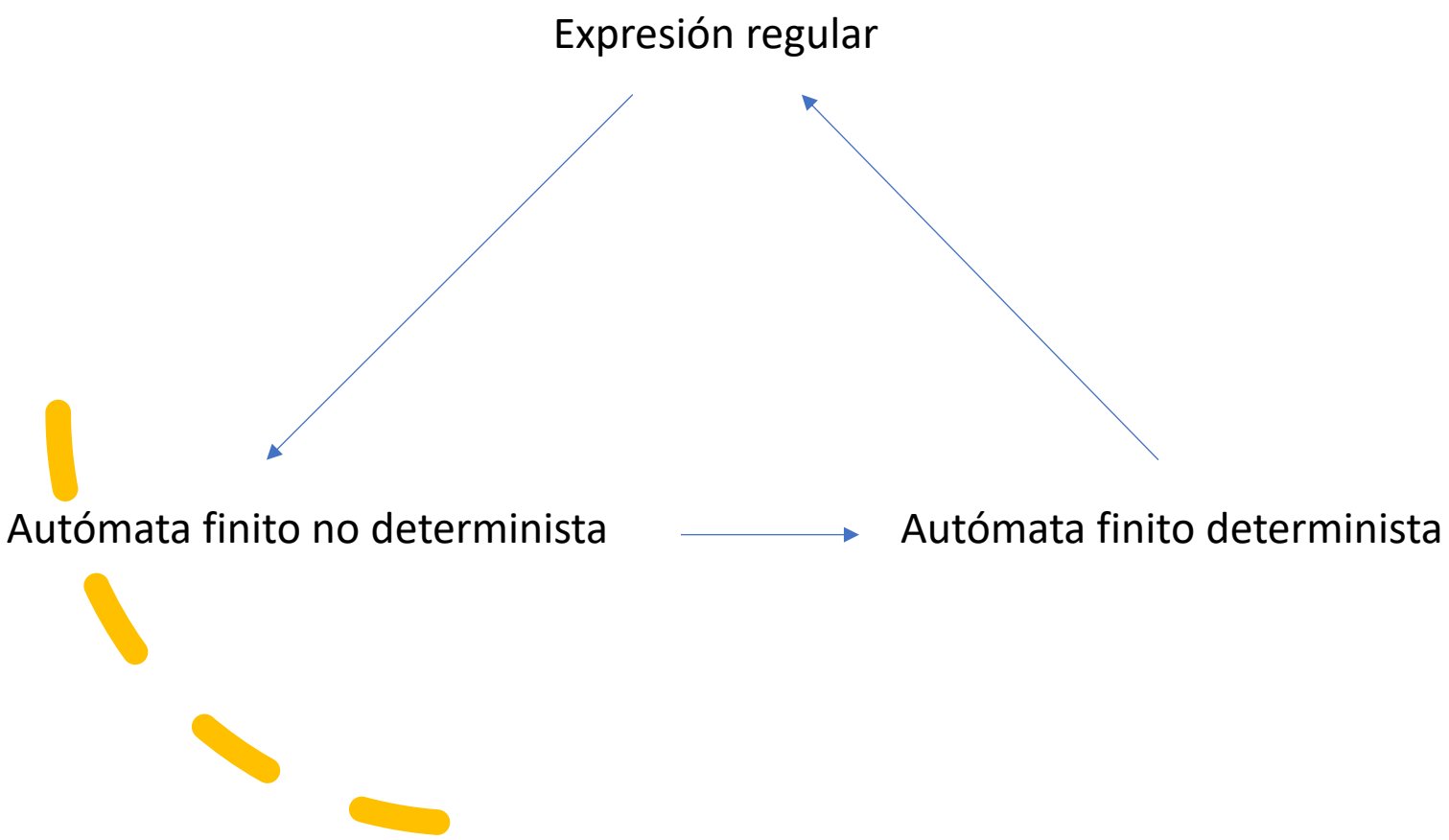
Expresiones regulares
en la vida real

- Prof. Adrián Lara
- Modificado por Prof. Maureen Murillo
- Teoría de la Computación
- Escuela de Computación e Informática
- Universidad de Costa Rica

Teoría vs práctica

Teoría



Práctica

- Lookahead
- Lookbehind
- Lookaround
- Backreferences
- Lazy quantifiers
- Greedy quantifiers

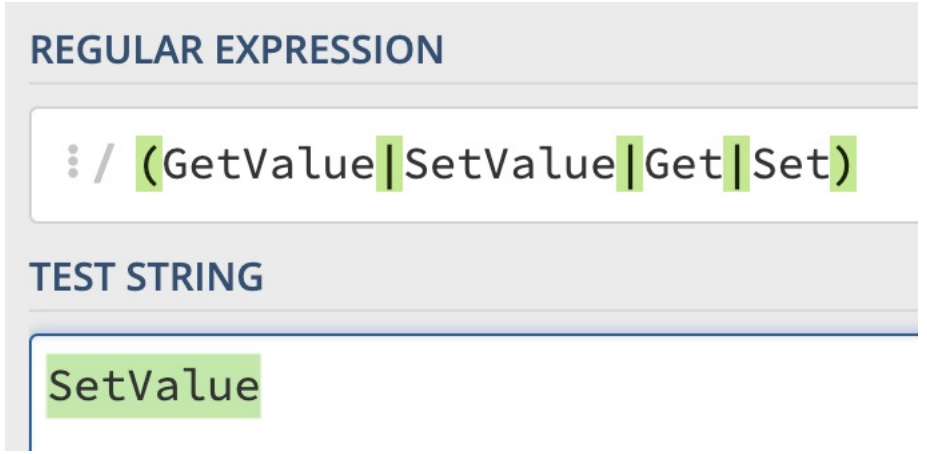
En la práctica, los *engines* de regex de la mayoría de lenguajes soportan mucho más que lenguajes regulares

Información útil de regex más allá de los comandos básicos

- Comandos que agilizan la forma en que especificamos una regex
  - \w (word character): [A-Za-z0-9_]
  - \d (digit): [0-9]
  - \s (whitespace character): [\t\r\n\f]
  - \W, \D, \S:
    negación de los anteriores [^\w], etc.

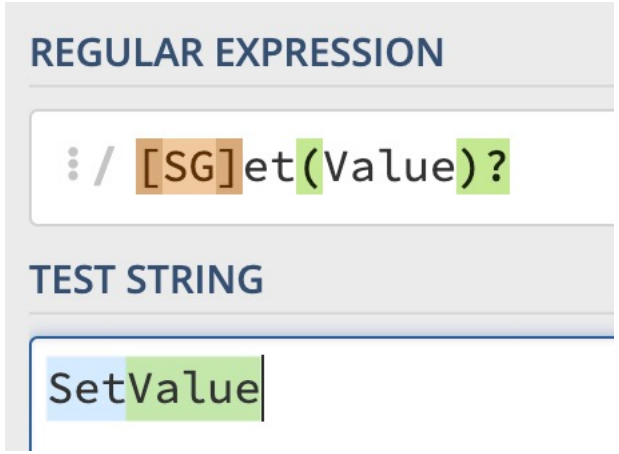
Ejemplos de situaciones usando engines

Por defecto, un regex *engine* es ansioso o ***eager***



Sugerencias:

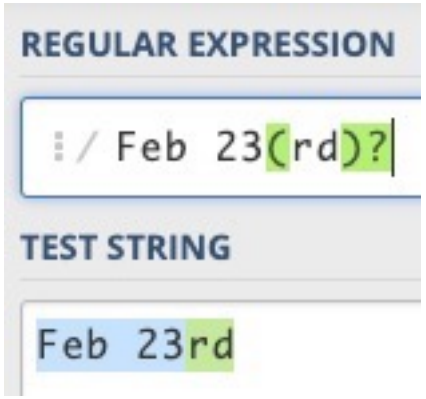
- Tomar en cuenta el orden (más probables antes, más largas antes)
- Factorizar

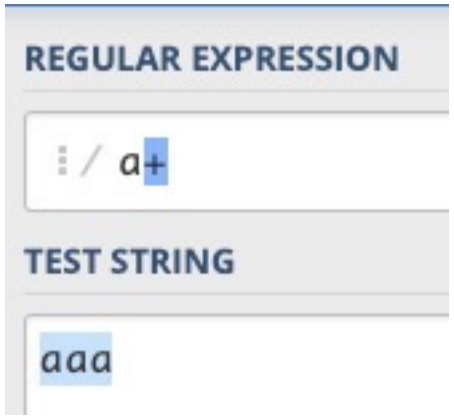


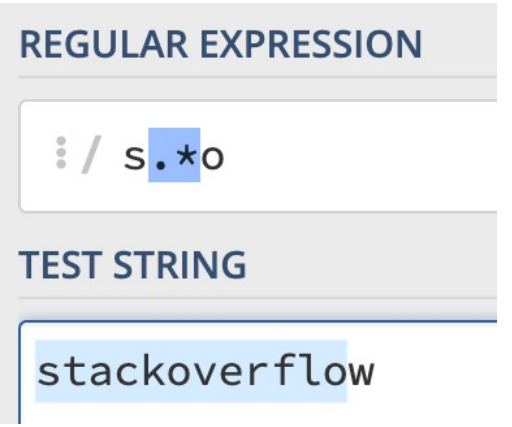
Ejemplos de situaciones usando *engines*

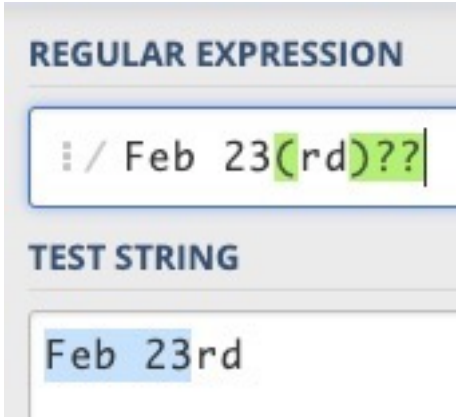
Por defecto, los operadores cuantificadores (+, *, ?) son egoístas o *greedy.* Buscan capturar la mayor cantidad posible de caracteres.

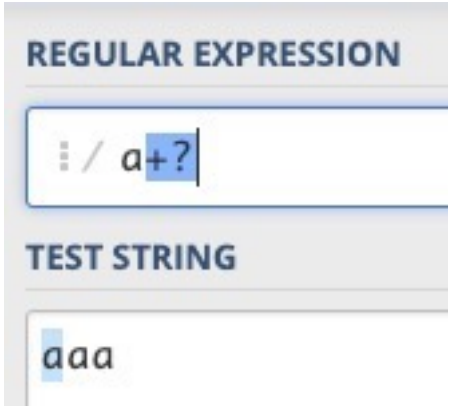
Se puede agregar un ? para que el operador sea vagabundo o lazy

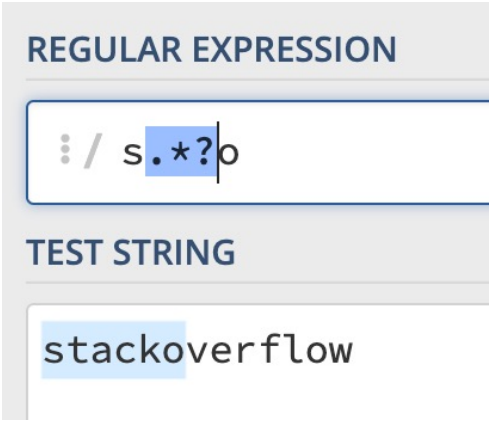












Operadores *greedy*

Greedy quantifier	Lazy quantifier	Description
*	*?	Star Quantifier: 0 or more
+	+?	Plus Quantifier: 1 or more
?	??	Optional Quantifier: 0 or 1
{n}	{n}?	Quantifier: exactly n
{n,}	{n,}?	Quantifier: n or more
{n,m}	{n,m}?	Quantifier: between n and m

> Add a ? to a quantifier to make it ungreedy i.e lazy.

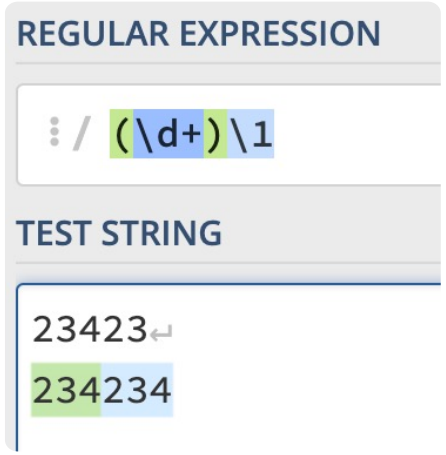
Example:
test string : *stackoverflow*
greedy reg expression : `s.*o` output: **stackoverflow**
lazy reg expression : `s.*?o` output: **stacko**verflow

Backreferences

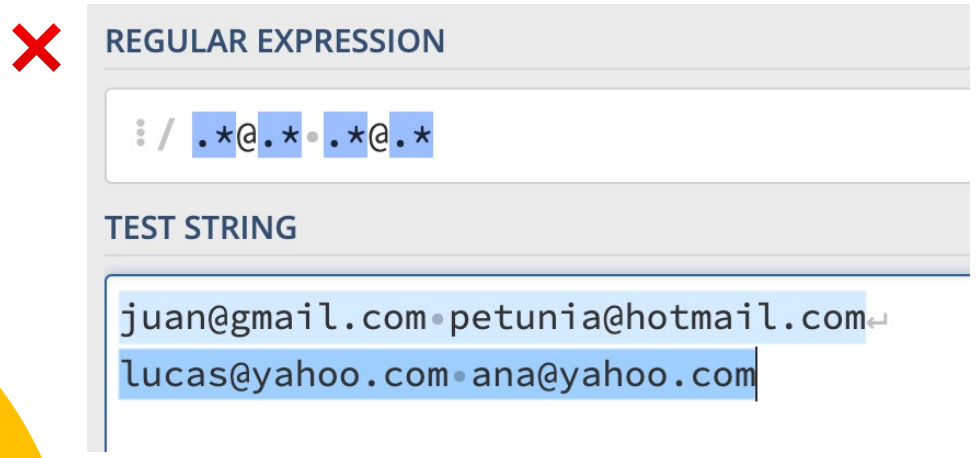
- Es práctico poder repetir un *match* exacto de parte de una expresión

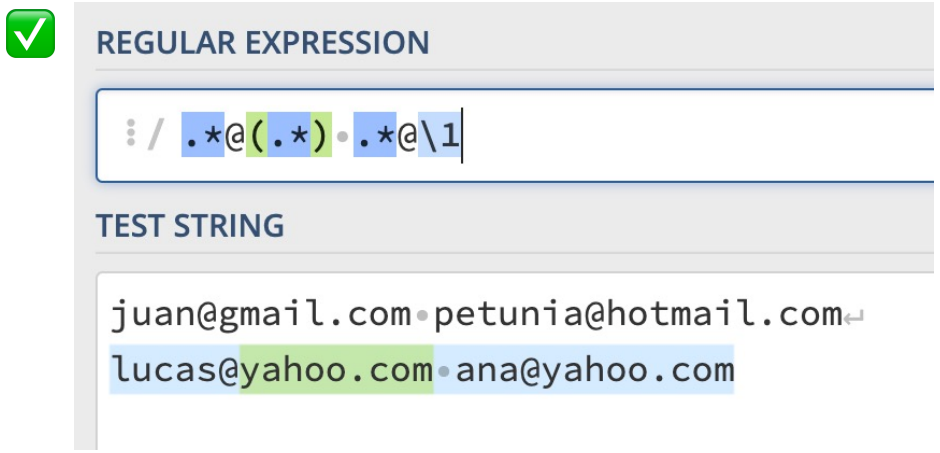
Ejemplo 1: observe que los paréntesis rodean la parte de la expresión que hace match de \d+. Ese lenguaje se podría describir como “dos secuencias idénticas de dígitos”, o “dos veces la misma hilera de dígitos”.

¡Ese lenguaje NO es regular!



Ejemplo 2: regex que evalúe dos direcciones de correo con dominio igual.





Lookahead y lookbehind

- Operaciones que hacen match sin consumir ningún caracter de la entrada

Operación	Sintaxis	Explicación
Positive lookahead	q(?=u)	Haga match de la q únicamente si después viene una u
Negative lookahead	q(?!u)	Haga match de la q únicamente si después no viene una u
Positive lookbehind	(?<=u)q	Haga match de la q únicamente si antes viene una u
Negative lookbehind	(?<!u)q	Haga match de la q únicamente si antes no viene una u

Nota: se puede usar un lookahead o lookbehind sin hacer match de nada, sino simplemente para validar una condición. Es decir, las q del cuadro anterior son opcionales. (Ver siguiente filmina)

Validación de contraseña usando lookahead

- Requerimientos
  - Entre 6 y 10 caracteres word \w
  - Al menos una minúscula [a-z]
  - Al menos tres mayúsculas [A-Z]
  - Al menos un dígito \d

Estrategia a seguir: usar un lookahead para validar cada condición (veremos al final que con solo n-1 lookaheads es suficiente)

Referencia con explicación detallada del ejemplo: https://www.rexegg.com/regex-lookarounds.html

Solución (1)

- Cada requerimiento por su cuenta es fácil de revisar:
  - Entre 6 y 10 caracteres word: \w
  - Al menos una minúscula: [a-z]
  - Al menos tres mayúsculas: [A-Z]
  - Al menos un dígito \d

El problema es que si hacemos todas las posibles combinaciones de orden, es muy complicado.

Solución (2)

- ¿Cómo se vería un lookahead para cada requerimiento?
  - Entre 6 y 10 caracteres word \w — \A\w{6,10}\z
  - Al menos una minúscula [a-z] — [^a-z]*[a-z]
  - Al menos tres mayúsculas [A-Z] — ([^A-Z]*[A-Z]){3}
  - Al menos un dígito \d — \D*\d

Mejor verificar que se cumplen las cuatro condiciones antes de capturar caracteres:

(?=\A\w{6,10}\z)

(?=[^a-z]*[a-z])

(?=([^A-Z]*[A-Z]){3})

(?=\D*\d)

Solución (3)

Primera solución

(?=\A\w{6,10}\z)(?=[^a-z]*[a-z])(?=([^A-Z]*[A-Z]){3})(?=\D*\d).*

Solución mejorada

(?=[^a-z]*[a-z])(?=([^A-Z]*[A-Z]){3})(?=\D*\d)\A\w{6,10}\z

Esta expresión regular acepta únicamente aquellas hileras que cumplan con los tres lookaheads y con el patrón descrito por \A\w{6,10}\z.

Para revisar n condiciones se necesitan n-1 lookaheads/lookbehinds.

¿Cómo escribir un lookahead/lookbehind?

No es necesario que la regex del lookahead/lookbehind haga *match* con toda la hilera. Basta con que la expresión del lookahead/lookbehind busque lo estrictamente lo necesario, pero que no tenga conflictos con el resto de la hilera.

Por ejemplo, si se desea validar cualquier hilera que al menos tenga un dígito en cualquier posición, la regex **\D*\d** no sirve. Opciones correctas serían: **^(\D*\d\D*)*$** o **^.*\d.*$** .

Sin embargo, podemos escribir una versión de regex con lookahead que incluya **\d**, tal como: **^.*(?=\d).*$** , que significa que haya al menos un dígito.

Backtracking

¿Notaron en esta presentación patrones como:

\D*\d, o bien [^A-Z][A-Z] ?

¿Por qué es mejor \D*\d que .*\d ?

Los *engines* de regex usan backtracking para probar cuál camino lleva a la aceptación de una hilera (algo parecido a lo que hacemos a pie con un autómata finito no determinista).

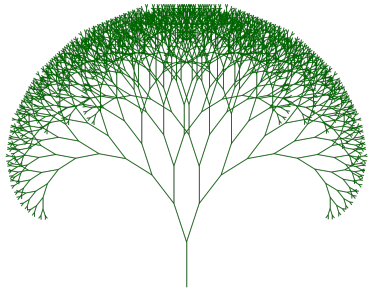
Backtracking (2)

Si la regex es **.*\d** y la entrada es **abcd1**, el **.*** puede capturar el **1** y es lo que va a ocurrir por defecto. Sin embargo, se terminará la entrada y faltará aún validar un dígito, por lo que esta hilera no sería aceptada.

Entonces, el *engine* descarta esa opción y retrocede para ver si otro camino era posible. Así es como intenta que **.*** únicamente capture **abcd** y que **\d** capture el **1**.

Bactracking es una herramienta poderosa que permite que los *engines* sean más flexibles y más eficientes para reconocer hileras.

Pero:



Backtracking catastrófico

Suponga que se usa la siguiente regex: **^(A+)*B**

Si intentamos reconocer la hilera AAAC:

1. Primer caso: el operador A+ consume todos los caracteres. Fallo y backtrack por la B.
2. Segundo caso: el operador A+ suelta la última A y el operador * consume la última A. Fallo y backtrack por la B.
3. Tercer caso: el operador A+ suelta dos A's y el proceso se repite…

A+	A+	A+
AAA	–	–
AA	A	–
AA	–	–
A	AA	–
A	A	A
A	A	–
A	–	–
–	–	–

Fuente del ejemplo usado: https://www.rexegg.com/regex-explosive-quantifiers.html

Backtracking catastrófico

- Si no somos cuidadosos a la hora de escribir una expresión regular, podemos convertir el proceso de match en algo de complejidad $O(2^n)$
- Intente por su cuenta este ejemplo.

Number of *As*	Steps to Fail
1, e.g. *AC*	7
2, e.g. *AAC*	14
3, e.g. *AAAC*	28
4	56
5	112
10	3,584
20	3,670,016 — RegexBuddy has given up. How about your program?
100	4,436,777,100,798,802,905,238,461,218,816

Síntomas de backtracking catastrófico

- Fuente y explicación detallada aquí

Síntoma	Posible solución
Un cuantificador dentro de otro (A+)*	Impedir backtracking. Convertir el cuantificador en atómico o en posesivo
Cuantificadores contiguos no son mutuamente excluyentes ^\d+\w*@	Usar contrastes (\D*\d* es mejor que .*\d*
Dos o más alternativas no son mutualmente excluyentes ^(?:\d|\w)+@	Factorizar. a+(b|c)+ es mejor que (a+b+|a+c+) Como mínimo, usar [ ] que son atómicos

Cuantificadores posesivos

Cuantificador greedy que no produce backtracking

*+	Consume 0 ó más veces y no retrocede.
++	Consume 1 ó más veces y no retrocede.
?+	Consume 0 ó 1 vez y no retrocede.
{n}+	Consume exactamente n veces y no retrocede.
{n,}+	Consume al menos n veces y no retrocede.
{n,m}+	Consume al menos n veces y no más de m y no retrocede.

Por ejemplo, la expresión regular **a++a** no reconoce la hilera **aaaa** porque el **a+** consume las cuatro **a** y no hace backtracking

Interoperabilidad

Último mensaje: para usar aspectos “avanzados” de expresiones regulares, es conveniente estudiar cada *engine* por separado (Java, Javascript, AWS, Python, Perl…). No todos tienen los mismos comandos.

Fuentes valiosas para aprender más

https://www.regular-expressions.info/tutorial.html

Gran parte de esta presentación se basa en los ejemplos de esta página