Programming Languages and Computation

Week 2: Regular Expressions

* 1. For each of the following regular expressions over $\{0, 1\}$, write it out in the official syntax with all of the implicit parentheses and the concatenation operators tediously put back in (according to our agreed conventions).

(a) $(00^* + 11^*)^*$

(b) $(01 + 10)(01 + 10)$

(c) $(0 + 1(01^*0)^*1)^*$

Solution

(a) $(((0 \cdot (0^*)) + (1 \cdot (1^*)))^*)$

(b) $(((0 \cdot 1) + (1 \cdot 0)) \cdot ((0 \cdot 1) + (1 \cdot 0)))$

(c) $((0 + ((1 \cdot (((0 \cdot (1^*)) \cdot 0)^*)) \cdot 1))^*)$

* 2. For each of the following regular expressions over $\{a, b\}$, write it out with the minimum number of parentheses and all concatenation operators suppressed (according to our agreed conventions).

(a) $(((a + \epsilon) \cdot (b + \epsilon))^*)$

(b) $(((a + b) \cdot (b^*))^*)$

(c) $(((a^*) + (((a \cdot b) \cdot a)^*))^*)$

Solution

(a) $((a + \epsilon)(b + \epsilon))^*$

(b) $((a + b)b^*)^*$

(c) $(a^* + (aba)^*)^*$

* 3. Consider the following pairs of regular expressions over $\{a, b\}$. For each, find a word that is matched by the first, but not by the second.

(a)	$(a+b)^*$	$a^* + b^*$
(b)	$(a^*b)^*$	$(ab^*)^*$
(c)	$\emptyset^*$	$(a+b)$

Solution

(a) Any words involving a and b together, such as ab

(b) b

(c) ϵ (recall that, for any regex R, R^* matches the empty string)

* 4. Give traces to show that the regular expression $(ab+a)^*$ matches each of the following strings:

(a) ϵ

(b) aab

(c) $abab$

Solution

Out of laziness I will typically omit ϵ labels, you can do too.

(a)

$$(ab+a)^* \longrightarrow \epsilon$$

(b)

$$\begin{aligned} (ab+a)^* &\longrightarrow (ab+a)(ab+a)^* \\ &\longrightarrow a(ab+a)^* \\ &\xrightarrow{a} (ab+a)^* \\ &\longrightarrow (ab+a)(ab+a)^* \\ &\longrightarrow ab(ab+a)^* \\ &\xrightarrow{a} b(ab+a)^* \\ &\xrightarrow{b} (ab+a)^* \\ &\longrightarrow \epsilon \end{aligned}$$

(c)

$$\begin{aligned}
(ab+a)^* &\longrightarrow (ab+a)(ab+a)^* \\
&\longrightarrow ab(ab+a)^* \\
&\xrightarrow{a} b(ab+a)^* \\
&\xrightarrow{b} (ab+a)^* \\
&\longrightarrow (ab+a)(ab+a)^* \\
&\longrightarrow ab(ab+a)^* \\
&\xrightarrow{a} b(ab+a)^* \\
&\xrightarrow{b} (ab+a)^* \\
&\longrightarrow \epsilon
\end{aligned}$$

** 5.

(a) Draw a directed graph with the following components:

- The nodes/vertices are: $\{R \mid \exists w \in \Sigma^*.\ (a^* + b)^* \xrightarrow{w}{}^* R\}$
- There is an ℓ-labelled edge between vertex R and vertex S just if $R \xrightarrow{\ell} S$.

(b) Give infinitely many different traces to show that the regular expression $(a^* + b)^*$ matches the string ab.

(You do not have enough paper to enumerate them explicitly, so please just indicate the general pattern.)

Solution

(a) Here is the graph:

$$\begin{tikzcd}
 & \epsilon & \\
 & (a^*+b)^* \arrow[u, "\epsilon"] \arrow[d, "\epsilon"] & \\
 & (a^*+b)(a^*+b)^* \arrow[dl, "\epsilon"'] \arrow[dr, "\epsilon"] & \\
a^*(a^*+b)^* \arrow[uur, bend left=60, "\epsilon"] \arrow[d, bend right, "\epsilon"'] & & b(a^*+b)^* \arrow[uul, bend right=60, "b"'] \\
aa^*(a^*+b)^* \arrow[u, bend right, "a"'] & &
\end{tikzcd}$$

(b) For each natural number i we can construct a (prefix) of the trace in which the following subtrace is repeated i-times:

$$(a^* + b)^* \longrightarrow (a^* + b)(a^* + b)^* \longrightarrow a^*(a^* + b)^* \longrightarrow (a^* + b)^*$$

Then this is followed by a fixed suffix that actually matches the word:

$$\begin{aligned}
&\longrightarrow (a^* + b)(a^* + b)^* \\
&\longrightarrow a^*(a^* + b)^* \\
&\longrightarrow aa^*(a^* + b)^* \\
&\xrightarrow{a} a^*(a^* + b)^* \\
&\longrightarrow (a^* + b)^* \\
&\longrightarrow (a^* + b)(a^* + b)^* \\
&\longrightarrow b(a^* + b)^* \\
&\xrightarrow{b} (a^* + b)^* \\
&\longrightarrow \epsilon
\end{aligned}$$

In other words, we are going round the ϵ-labelled loop in the graph as many times as we like, before taking the left and then right branches, which actually match ab.

** 6. Design regular expressions to match the following strings over $\{a, b\}$:

(a) All strings that contain an even number of a

(b) All strings that contain an odd number of b

(c) All strings that contain either an even number of a or an odd number of b

Solution

(a) $b^* + (b^*ab^*ab^*)^+$ – a string containing an even number of a can be decomposed into a non-empty concatenation of substrings each of which contains two a or, is all bs

(b) $(a^*ba^*)(a^*ba^*ba^*)^*$ – a string containing an odd number of b can be decomposed into an initial substring containing exactly one b followed by a concatenation of substrings each of which contains two b.

(c) $b^* + (b^*ab^*ab^*)^* + (a^*ba^*)(a^*ba^*ba^*)^*$

** 7. Design regular expressions for the following programming language lexemes over the ASCII alphabet. You will find it convenient to define abbreviations to help present the expressions compactly.

(a) A *C program identifier* is any string of length at least 1 containing only letters ('a'–'z', lower and uppercase), digits ('0'–'9') and the underscore, and which begins with a letter or the underscore.

(b) In most programming languages, *integer literals* can be written in:

- decimal – as a non-empty sequence of digits
- hexadecimal – as a non-empty sequence of characters from '0'–'9','a'–'e' (upper or lowercase) that are preceded by "0x"
- binary – as a non-empty sequence of bits '0' and '1' that are preceded by "0b"

Solution

(a) $R(R+S)^*$ where $R = [abc\dots xyzABC\dots XYZ\_]$ and $S = [0123456789]$

(b) $R^+ + 0x(R + [abcdefABCDEF])^+ + (0b[01]^+)$ where $R = [0123456789]$

** 8. Design regular expressions to match the following sets of strings over the alphabet of ASCII characters. You will find it convenient to define some abbreviations to help present the expressions compactly.

(a) Valid Bristol University usernames

(b) Valid 24 hour clock times in format HH:MM

(c) Valid IPv4 addresses written in decimal

Solution

(a) $RRSSSSS$ where $R = [abcdefghijklmnopqrstuvwxyz]$ and $S = [0123456789]$

(b) R:S where $R = [01][0123456789] + 2[0123]$ and $S = [012345][0123456789]$

(c) $R.R.R.R$ where $R = S + SS + ([01]SS + 2[01234]S + 25[012345])$ and $S = [0123456789]$ – here the idea is that the three disjuncts of R represent octets consisting of 1, 2 or 3 digits respectively.

*** 9. Use traces to argue that $(a^*b^*)^*$ can match all strings over $\{a, b\}$.

Hint: Consider what can be matched using a trace that both starts and ends in $(a^*b^*)^*$.

Solution

First observe that $(a^*b^*)^* \xrightarrow{\epsilon} \epsilon$. Then, observe that $(a^*b^*)^* \xrightarrow{a}{}^* (a^*b^*)^*$ by:

$$\begin{aligned}(a^*b)^* &\longrightarrow a^*b^*(a^*b^*)^* \\ &\longrightarrow aa^*b^*(a^*b^*)^* \\ &\xrightarrow{a} a^*b^*(a^*b^*)^* \\ &\longrightarrow b^*(a^*b^*)^* \\ &\longrightarrow (a^*b^*)^*\end{aligned}$$

Similarly, observe that $(a^*b^*)^* \xrightarrow{b}{}^* (a^*b^*)^*$ by:

$$\begin{aligned}(a^*b)^* &\longrightarrow a^*b^*(a^*b^*)^* \\ &\longrightarrow b^*(a^*b^*)^* \\ &\longrightarrow bb^*(a^*b^*)^* \\ &\xrightarrow{b} b^*(a^*b^*)^* \\ &\longrightarrow (a^*b^*)^*\end{aligned}$$

By definition, a word over $\{a, b\}$ is a sequence of a and b. Hence, for any such word, we can construct a trace that shows the word is matched by glueing together copies of the above subtraces, with one copy per letter, and one final descent into ϵ.

*** 10. Give regular expressions to match each of the following sets of strings over $\{a, b\}$:

(a) Strings that do not contain the substring a

(b) Strings that do not contain the substring ab

(c) Strings that do not contain the substring aba

Solution

(a) b^*

(b) b^*a^*

(c) $b^* + b^*a(a + bb^+a)^*b^*$

This requires some justification. First, notice that this regex cannot possibly match any string that contains aba as a substring, because the only place that b can occur between two a is as part of bb^+a in which case there are at least two b.

However, to see that it matches all the strings we need requires a bit more thought. Clearly, we can match a string containing no a. Now consider strings v that do contain an a. Such strings can be cut into pieces $v = u_1u_2 \cdots u_n \cdot b^*$ such that each piece u_i is of shape b^*a (i.e. contains exactly one a at the end) and then there are possibly some b at the very end of the string. For example, $bbbabbbaaaaabbabbab$ can be divided as:

$$bbba|bbba|a|a|a|a|bba|bba|b$$

Since v contains an a, v contains at least one of these pieces u_1, and for every subsequent piece u_i of shape b^*a (if there are any) it must be that either there are no b, so $u_i = a$ or there are at least two b, so $u_i = bb^+a$. Hence, we arrive at the expression above.