Programming Languages and Computation

Week 3: Regular Expressions

* 1. For each of the following regular expressions over $\{0, 1\}$, draw the expression explicitly as a tree.

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

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

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

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

* 3. Give proof trees/derivations to show that the regular expression $(ab + a)^*$, over $\{a, b\}$, matches each of the following strings:

(a) ϵ

(b) ab

(c) aab

* 4. Give proof trees/derivations to show that the regular expression 0^*10^*, over $\{0, 1\}$ matches the following strings:

(a) 1

(b) 01

(c) 001

* 5. 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. You need not justify your answer.

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

(b) $(a^* b)^*$ $\quad$ $(a b^*)^*$

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

** 6. Design regular expressions to match the following strings over $\{a, b\}$. In general, if asked to design or construct regexes, you need not justify your answer.

(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

** 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"

** 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

** 9. Make sure you have completed at least parts 0-2 of the Turtle compiler project: https://github.com/uob-coms20007/turtle

*** 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