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.

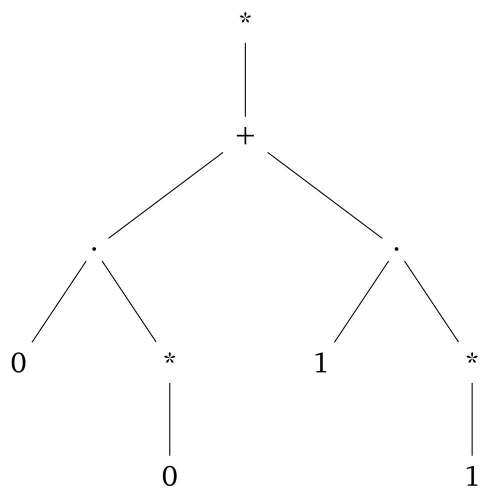
(a) $(00^* + 11^*)^*$

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

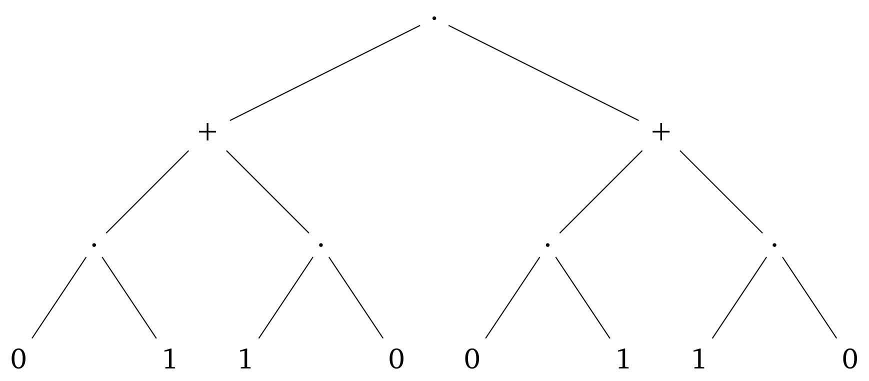
(c) $(0 + 1(01^*0)^*1)^*$

Solution

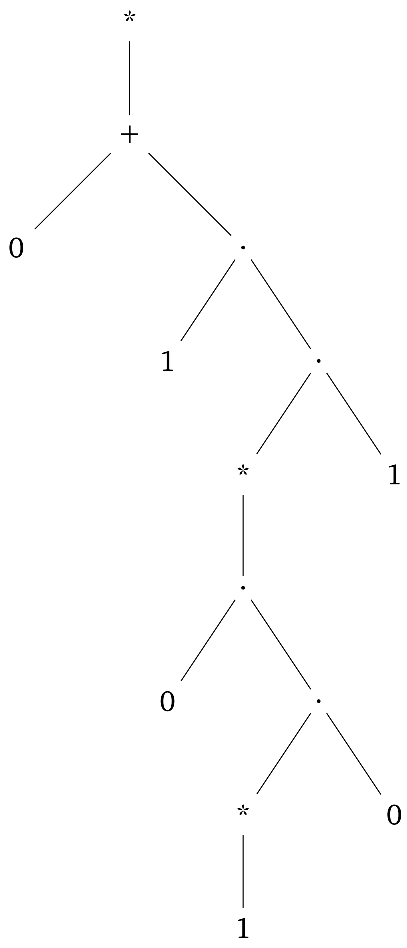
(a)



(b)



(c)



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

(a) $\epsilon$

(b) $ab$

(c) $aab$

Solution

You don't need to write the rule names on the tree, but I will to make it easier for you to see what is going on.

(a)

$$\text{(StarB)}\ \dfrac{}{(ab+a)^* \text{ matches } \epsilon}$$

(b)

$$\text{(StarS)}\ \dfrac{\text{(ChoiceL)}\ \dfrac{\text{(Concat)}\ \dfrac{\text{(Char)}\ \dfrac{}{a \text{ matches } a} \quad \text{(Char)}\ \dfrac{}{b \text{ matches } b}}{ab \text{ matches } ab}}{ab+a \text{ matches } ab} \qquad \text{(StarB)}\ \dfrac{}{(ab+a)^* \text{ matches } \epsilon}}{(ab+a)^* \text{ matches } ab}$$

(c) Here I abbreviate the tree given in the previous answer as (b):

$$\text{(StarS)}\ \dfrac{\text{(ChoiceR)}\ \dfrac{\text{(Char)}\ \dfrac{}{a \text{ matches } a}}{ab+a \text{ matches } a} \qquad \text{(b)}\ \dfrac{\vdots}{(ab+a)^* \text{ matches } ab}}{(ab+a)^* \text{ matches } aab}$$

* 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

Solution

(a) $0^*10^*$ is a concatentation of three regexes: $0^*$, 1 and $0^*$, so we need to split the target string into three pieces, each of which is matched by the corresponding regex. So we split 1 as $\epsilon\,1\,\epsilon$, with the first $\epsilon$ matched by $0^*$, the 1 matched by the regex 1 and the second $\epsilon$ matched by the last $0^*$.

$$\text{(Concat)}\ \dfrac{\text{(StarB)}\ \dfrac{}{0^* \text{ matches } \epsilon} \qquad \text{(Concat)}\ \dfrac{\text{(Char)}\ \dfrac{}{1 \text{ matches } 1} \quad \text{(StarB)}\ \dfrac{}{0^* \text{ matches } \epsilon}}{10^* \text{ matches } 1}}{0^*10^* \text{ matches } 1}$$

(b)

$$\text{(Concat)}\,\dfrac{\text{(StarS)}\,\dfrac{\text{(Char)}\,\dfrac{}{0 \text{ matches } 0} \qquad \text{(StarB)}\,\dfrac{}{0^* \text{ matches } \epsilon}}{0^* \text{ matches } 0} \qquad \text{(Concat)}\,\dfrac{\text{(Char)}\,\dfrac{}{1 \text{ matches } 1} \qquad \text{(StarB)}\,\dfrac{}{0^* \text{ matches } \epsilon}}{10^* \text{ matches } 1}}{0^*10^* \text{ matches } 01}$$

(c) Here I abbreviate the proof tree for $10^*$ matches 1, which is a subtree of the previous answer, as $\vdots$:

$$\text{(Concat)}\,\dfrac{\text{(StarS)}\,\dfrac{\text{(Char)}\,\dfrac{}{0 \text{ matches } 0} \qquad \text{(StarB)}\,\dfrac{\text{(Char)}\,\dfrac{}{0 \text{ matches } 0} \qquad \text{(StarB)}\,\dfrac{}{0^* \text{ matches } \epsilon}}{0^* \text{ matches } 0}}{0^* \text{ matches } 00} \qquad \dfrac{\vdots}{10^* \text{ matches } 1}}{0^*10^* \text{ matches } 001}$$

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

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

Solution

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

(b) $b$

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

** 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 $b$

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 $b$s

(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 $R = [abc\ldots xyzABC\ldots XYZ\_]$ 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. 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 $aba$

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.