2.4

To show that (R|S)\* = (R\*S\*)\*, where R and S are regular expressions, we need to prove that any string in (R|S)\* is also in (R\*S\*)\*, and vice versa.

1. (R|S)\* ⊆ (R\*S\*)\*:

Let w be a string in (R|S)\*. This means w is a concatenation of strings, each of which is either in R or in S. We want to show that w is also in (R\*S\*)\*.

We can write w as a concatenation of substrings w₁, w₂, ..., wₙ where each wᵢ is either in R or in S:

w = w₁w₂...wₙ

We can split wᵢ into two parts: wᵢ = xᵢyᵢ, where xᵢ is the part of wᵢ that belongs to R and yᵢ is the part that belongs to S. We can then rewrite w as:

w = (x₁y₁)(x₂y₂)...(xₙyₙ)

Now, xᵢ is in R, and yᵢ is in S. Hence, xᵢ is in R and yᵢ is in S\*.

w = (x₁ · S\*)(x₂ · S\*)...(xₙ · S\*)

Now, we have a concatenation of strings, each of which is of the form xᵢ · S\*, where xᵢ is in R and S\* is in S\*. This is precisely the definition of (R · S\*)\*.

Therefore, w is also in (R · S\*)\*.

2. (R · S\*)\* ⊆ (R|S)\*:

Now let's prove the other direction, i.e., that (R · S\*)\* ⊆ (R|S)\*.

Let w be a string in (R · S\*)\*. This means w is a concatenation of strings, each of which is of the form x · S\*, where x is in R. We want to show that w is also in (R|S)\*.

Since x is in R and S\* is in S\*, w is in (R|S)\*.

Therefore, (R|S)\* = (R · S\*)\*.

This completes the proof.

2.5

Certainly! To demonstrate that the concatenation of regular expressions, denoted as RS, is not necessarily equal to SR, let's provide a counterexample.

Let:

- R represent the regular expression "a\*", which denotes zero or more occurrences of 'a'.

- S represent the regular expression "b\*", which denotes zero or more occurrences of 'b'.

Now, let's evaluate RS (concatenation of R and S):

RS = (a\*)(b\*)

This regular expression RS represents zero or more 'a's followed by zero or more 'b's.

Similarly, let's evaluate SR (concatenation of S and R):

SR = (b\*)(a\*)

This regular expression SR represents zero or more 'b's followed by zero or more 'a's.

Now, let's demonstrate that RS is not equal to SR by providing a counterexample:

Consider the string "ab":

- "ab" belongs to RS, as it matches zero or more 'a's followed by zero or more 'b's: (a\*)(b\*).

- However, "ab" does not belong to SR, as it does not match zero or more 'b's followed by zero or more 'a's: (b\*)(a\*).

Therefore, RS ≠ SR, and we have shown this with the counterexample "ab."

2.6

To show that (R|S)\* is not necessarily equal to (R\*|S\*), let's provide a counterexample.

Let:

- R represent the regular expression "a\*", which denotes zero or more occurrences of 'a'.

- S represent the regular expression "b\*", which denotes zero or more occurrences of 'b'.

Now, let's evaluate (R|S)\*:

(R|S)\* = (a\*|b\*)\*

This regular expression represents zero or more occurrences of 'a's or 'b's.

Next, let's evaluate (R\*|S\*):

(R\*|S\*) = (a\*|b\*)

This regular expression represents zero or more occurrences of 'a' or zero or more occurrences of 'b'.

Now, let's provide a counterexample to show that (R|S)\* is not equal to (R\*|S\*):

Consider the string "ab":

- "ab" belongs to (R|S)\*, as it matches zero or more 'a's or zero or more 'b's: (a\*|b\*).

- However, "ab" does not belong to (R\*|S\*), as it does not match zero or more 'a's or zero or more 'b's: (a\*|b\*).

Therefore, (R|S)\* ≠ (R\*|S\*), and we have shown this with the counterexample "ab."

2.7

In the regular expression R = (ab|b)\*c, the language L(R) represents strings that start with zero or more occurrences of "ab" or "b" followed by a "c". Let's determine which of the given strings are in L(R):

1. ababbc

- This string is in L(R) because it starts with "abab" (or "b") and ends with "c".

2. c

- This string is in L(R) because it directly ends with "c".

3. babc

- This string is not in L(R) because it doesn't start with "ab" or "b".

4. abab

- This string is not in L(R) because it doesn't end with "c".

Therefore, the strings that are in L(R) are ababbc and c.

2.8

In the regular expression R = ab\*c(a|b)c, the language L(R) represents strings that start with "a", followed by zero or more "b"s, then a "c", and finally either an "a" or a "b", and ends with "c". Let's determine which of the given strings are in L(R):

1. acac

- This string is in L(R) because it follows the pattern: "a", zero "b"s, "c", "a", "c".

2. acbbbc

- This string is in L(R) because it follows the pattern: "a", three "b"s, "c".

3. abcac

- This string is not in L(R) because it doesn't follow the pattern of the regular expression.

4. abcc

- This string is in L(R) because it follows the pattern: "a", zero "b"s, "c", "c".

Therefore, the strings that are in L(R) are acac, acbbbc, and abcc.

2.9

A regular expression that represents the language of all strings beginning and ending with "b" over the alphabet Σ = {a, b} can be expressed as follows:

R = b(Σ\*)b

Explanation:

- It starts with the character "b".

- (Σ\*) represents zero or more occurrences of any character from the alphabet (Σ = {a, b}).

- It ends with the character "b".

This regular expression will match strings like "bb," "bab," "babb," "bbb," etc., where "b" is at the beginning and end of each string.

2.10

To create a regular expression for strings whose length is a multiple of 3 over the alphabet Σ = {a, b}, we can use the concept of regular expressions for multiples of a particular length. Here's a regular expression that matches strings of length 3k, where k is a non-negative integer:

R = ((ΣΣΣ)\*)

Explanation:

- ΣΣΣ represents three consecutive characters from the alphabet Σ = {a, b}.

- (ΣΣΣ)\* allows for zero or more occurrences of three consecutive characters.

This regular expression will match strings whose length is a multiple of 3, such as "aaa," "bbb," "aabbaabb," "ababab," etc.

2.11

To create a regular expression for strings containing exactly 3 occurrences of '1' over the alphabet Σ = {0, 1}, we can use the regular expression concept along with the "at least" and "at most" approach. Here's the regular expression:

R = (0\*1)30\*

Explanation:

- 0\* represents zero or more occurrences of '0'.

- 1 represents a single '1'.

- (0\*1)3 ensures there are exactly 3 occurrences of '1' in the string.

- 0\* at the end allows for zero or more occurrences of '0' after the third '1'.

This regular expression will match strings containing exactly 3 occurrences of '1', with any number of '0's in between or after.

2.12

To create a regular expression for strings where every '1' is followed by at least one '0' over the alphabet Σ = {0, 1}, we can use the following regular expression:

R = (0\*10+)\*

Explanation:

- 0\* represents zero or more occurrences of '0'.

- 1 represents a single '1'.

- 0+ represents one or more occurrences of '0'.

- 0\*10+ ensures that every '1' is followed by at least one '0'.

- (0\*10+)\* allows for zero or more occurrences of this pattern in the string.

This regular expression will match strings where every '1' is followed by at least one '0'.

2.13