#1 Syntactic Sugar

Definition 1.52 of Sipser gives the formal definition of a regular expression. You might notice that this includes a minimal set of operators (namely union, concatenation, and star) compared to a real-world regular expression language. The set of additional operators found in these real-world languages are not essential; indeed, with a little bit of thought, we can use the minimal set to emulate these extended operators.

For each of the given extended regular expression operators, give an equivalent regular expression using the operators defined in Sipser.

1. R^+: matches one or more occurrences of R.
2. $R^?$: matches zero or one occurrences of R.
3. $[c_1 \cdots c_k]$: matches exactly one occurrence of any of the characters c_1, ..., c_k.
4. $n\{R\}$: matches n repetitions of R.
5. $R^\$$: matches any string that ends with R.

#2 Reading Regular Expressions

For each of the following languages, give two strings that are members and two strings that are *not* members. Assume the alphabet $\Sigma = \{1, 0\}$.

1. 0^*1^*
2. $0(10)^*1$
3. $0^* \cup 1^*$
4. $(\epsilon \cup 0)1$
5. $(111)^*$
6. $\Sigma^*0\Sigma^*1\Sigma^*0\Sigma^*$

#3 Creating Regular Expressions

For each of the following language, give a DFA/NFA and a regular expression. In all cases, $\Sigma = \{0, 1\}$. You may specify your DFA/NFA by diagram or by δ-table (either is fine).

1. $L_1 = \{\, w \mid w \text{ does not contain 01 or 101} \,\}$.
2. $L_2 = \{\, w \mid w \text{ starts with 0 and has odd length or starts with 1 and has even length} \,\}$.