1. Write a regular expression over the alphabet \{0, 1\} that describes a set \(S\) of all strings where each string has an odd number of 1’s and the 1’s in the string occur consecutively. Design a finite state machine that accepts set \(S\).

Solution:
The regular expression is \(0^* 1 (11)^* 0^*\)
The Finite State Machine that accepts set \(S\) is

![Finite State Machine Diagram]

2. Design a finite state machine over the alphabet \{a, b\} that accepts each string that ends with \(ab\).

Solution:
The finite state machine is:

![Finite State Machine Diagram for ab]
3. For each state i in the finite state machine in Problem 2, define a predicate $Q_i$ that defines every string $x$ of symbols "a" and "b" that can move the machine from its initial state to state i.

Solution:
$Q_1 = (x$ is the empty string$) \lor (x$ is the string b$) \lor (x$ ends with bb$)$

$Q_2 = (x$ ends with a$)$

$Q_3 = (x$ ends with ab$)$

4. Design a transducer $T$ whose input and output alphabets are \{0, 1\}. If the input string consists of only one symbol, $T$ does not output any string. If the input string $<S.1, S.2, .., S.k>$ has at least 2 symbols, then after receiving the i-th symbol $S.i$ from this string, where i is at least 2, $T$ outputs the symbol $S.(i-1)$ provided $S.i$ is different from $S.(i-1)$. For example, if the input string is $<0, 0, 1, 1, 1, 0, 1>$, then $T$ outputs the string $<0, 1, 0>$.

Solution:
The transducer $T$ is

5. Solve the following two regular expression equations with variables $P$ and $Q$:

\[
P = (10)^* \mid Q \\
Q = (10)P
\]

Simplify the regular expressions for $P$ and $Q$ as much as possible.

Solution:
Substitute $Q$ in $P = (10)^* \mid Q$.

$P = (10)^* \mid (10)P$

The solution for $P$ is $(\varepsilon \mid (10)(10)^*) \mid (10)^* = (10)^* \mid (10)^* = (10)^*$

So, $Q = (10)(10)^*$
6. Define a Haskell function `sumt` that takes as parameters integers `a`, `b`, and `n`, and computes, without using the infix operator \(^\)\, the sum of \(n\) terms \(T.1, T.2, \ldots, T.n\), where the \(i\)-th term is \(a \times (b^{(i-1)})\). (Hint: function `sumt` should call another function `term` that computes the \(i\)-th term \(T.i\) in the sequence of terms.)

Solution:
```haskell
term a b 1 = a
term a b i = (term a b (i-1)) * b
sumt a b 1 = term a b 1
sumt a b n = (sumt a b (n-1)) + term a b n
```

7. Define a Haskell function `min2` that takes as a parameter any list `xs` of two or more integers and computes a 2-tuple \((x, y)\), where `x` is the smallest integer in list `xs` and `y` is the second smallest integer in `xs`.

Solution:
```haskell
min2 [x,y] = (min x y, max x y)
min2 (x:xs)
    | x < m = (x,m)
    | x < n = (m,x)
    | True  = (m,n)
    where (m,n) = min2 xs
```