(a) Complete the implementation of `putvar` by supplying the right code to replace the dots in the receive clause in `var(X)`. **1 points**

(b) Draw a similar diagram to illustrate the messages passed during a call of `getvar(VarPid)`, which is intended to retrieve the current value of the variable. **1 points**

(c) Write the code for `getvar(VarPid)`, and the corresponding code that must be added to the body of `var(X)` to handle the messages that `getvar(VarPid)` sends. **2 points**

4. A *lock* can be implemented in Erlang as a server process which client processes communicate with to claim and release the lock. Assume we need to implement a *single* lock, which will be managed by a process registered under the name `lock`. Clients will call `claim()` and `release()` to claim and release the lock; in between these two calls we say that the client is *holding* the lock. If one client is holding the lock, then no other client may claim it until the first client releases the lock—calls of `claim()` in other clients should wait until the lock is released. Calls of `release()` in clients which are not holding the lock should be ignored.

(a) Draw a diagram to illustrate the message(s) a client and server should exchange when the client claims the lock. **1 points**

(b) Draw a diagram to illustrate the message(s) a client and server should exchange when the client releases the lock. **1 points**

(c) Write Erlang definitions of `claim()` and `release()`. **2 points**

(d) At any time, the server is in one of two states: either it is *unlocked*, or it is *locked by a particular client pid*. Define Erlang functions `unlocked()` and `locked(Pid)` that implement the behaviour of the server in the unlocked state, and locked-by-Pid state respectively. **2 points**