[Ex. 1] Two processes $p_1$ and $p_2$ want to access a single shared resource $r$. Consider the atomic propositions:

- $\text{req}_i$: holds when process $p_i$ is requesting access to $r$;
- $\text{use}_i$: holds when process $p_i$ has had access to $r$;
- $\text{rel}_i$: holds when process $p_i$ has released $r$.

with $i \in [1, 2]$. Use LTL formulas to specify the following properties:

1. mutual exclusion: $r$ is accessed by only one process at a time;
2. release: every time $p_1$ accesses $r$, it releases $r$ after some time;
3. priority: whenever both $p_1$ and $p_2$ require $r$, $p_1$ is granted access first;
4. no starvation: whenever $p_1$ requires $r$, it is eventually granted access.

[Ex. 2] Three dogs live in a house with two couches and a front garden. Let $\text{couch}_{i,j}$ represent the predicate “the dog $i$ sits on couch $j$” and $\text{garden}_i$ represent the predicate “the dog $i$ plays in the front garden”.

1. Write an LTL formula expressing the fact that whenever dog 1 plays in the garden then he keeps playing until he sits on some couch (but he may also play forever).
2. Write a CTL formula expressing the fact that dog 2 eventually plays in the garden whenever couch 1 is occupied by another dog.
3. Write a $\mu$-calculus formula expressing the fact that no more than one couch is occupied at any time by dog 3.

[Ex. 3] Given the $\mu$-calculus formula $\Phi = \mu x.((p \land \square x) \lor (\neg p \land \Diamond x))$ write its denotational semantics $[\Phi] \rho$ and evaluate it on the LTS below (where $V = \{s_1, s_2, s_3, s_4\}$ and $P = \{p\}$).

```
    s1 --p s2
    /   \          \
   /     \        /
  s3 --o s4
```
[Ex. 4] Write a GoogleGo function that takes one channel \texttt{ini} for receiving integers and one channel \texttt{ins} for receiving strings and returns a channel \texttt{outp} where all the messages received on \texttt{ini} and \texttt{ins} will be paired.

*Hint: define a \texttt{struct} to form pairs*

[Ex. 5] Write a GoogleGo function that takes two channels \texttt{f} and \texttt{q} and tries to send the stream of Fibonacci numbers on \texttt{f} but quits when it receives \texttt{true} on channel \texttt{q}. Write a \texttt{main} program to test the function by printing the first 10 Fibonacci numbers.

[Ex. 6] The \textit{asynchronous} \(\pi\)-calculus requires that outputs have no continuation:

\[
p := \texttt{nil} \mid \pi(y) \mid x(y).p \mid \tau.p \mid [x = y]p \mid p + p \mid p|p \mid (x)p \mid !p
\]

Show that any process in the original \(\pi\)-calculus can be represented in the asynchronous \(\pi\)-calculus using an extra (fresh) channel to simulate explicit acknowledgement of name transmission.

[Ex. 7] The \textit{polyadic} \(\pi\)-calculus allows communicating more than one name in a single action, i.e., its action prefixes are of the form:

\[
\pi := \tau \mid \pi(z_1,...z_n) \mid x(z_1,...z_n)
\]

The polyadic extension is useful especially when studying types for name passing processes. Show that the polyadic \(\pi\)-calculus can be encoded in the ordinary (monadic) \(\pi\)-calculus by passing the name of a private channel through which the multiple arguments are then passed in a sequence.