Coursework 2

1. The following expressions show the behaviour of an FTP server.

$$\mathbf{Client}(pid) \stackrel{\text{df}}{=} (\nu\, s)\overline{pid}\langle s\rangle.\bar{s}\langle userid, passwd\rangle.s \rhd \{\mathsf{sorry} : \ldots \;[\!]\; \mathsf{welcome} : \ldots\}$$

$$\mathbf{Init}(pid, nis) \stackrel{\text{df}}{=} \mathbf{FtpThread}\langle pid, nis\rangle \mid \mathbf{Authenticator}\langle nis\rangle$$

$$\mathbf{FtpThread}(pid, nis) \stackrel{\text{df}}{=} !pid(z).z(userid, passwd).(\nu\, s')\overline{nis}\langle s'\rangle.\bar{s'}\langle userid, passwd\rangle. \\ s' \rhd \{\mathsf{invalid} : z \lhd \mathsf{sorry}.\mathbf{0} \;[\!]\; \mathsf{valid} : z \lhd \mathsf{welcome}.\mathbf{Actions}\langle z\rangle\}$$

$$\mathbf{Actions}(s) \stackrel{\text{df}}{=} s \rhd \{\, \mathsf{get} : \mathbf{Actions}\langle s\rangle \;[\!] \\ \mathsf{put} : \mathbf{Actions}\langle s\rangle \;[\!] \\ \mathsf{bye} : \mathbf{0}\,\}$$

$$\mathbf{Authenticator}(nis) \stackrel{\text{df}}{=} !nis(x).x(userid, passwd).\,\mathbf{if}\ passwd = pw\ \mathbf{then}\ x \lhd \mathsf{valid}.\mathbf{0} \\ \mathbf{else}\ x \lhd \mathsf{invalid}.\mathbf{0}$$

where we assume that pw is the valid password of the user with $userid$.

   (a) Write a Go program to simulate the above processes.

      i. Complete the unspecified parts of the **Client** process, marked with '...'. For example, $\mathsf{sorry} : \mathbf{0}$.

      ii. Your implementation may not have the exact same behaviour as the above processes, but any difference must be justified and reasonable. You may not remove a part of the specification altogether, but you may change the behaviour slightly to account for language limitations.
      Hint: be careful with replication, the implementation should only create as many goroutines as needed.

   (b) You need to include at least a comments block that explain briefly how your program works at the beginning of the file. You may include more comments along your implementation to clarify the code, in particular to justify the changes of behaviour from the specification mentioned above.

   (c) Submit the code's go source file on CATe before the deadline.

2. Recall the synchronous and polyadic encodings of the π-calculus. Here, we will write $[\![P]\!]_{\text{sync}}$ for the synchronous encoding, and $[\![Q]\!]_{\text{poly}}$ for the polyadic encoding: P is a *synchronous* π-calculus term, and $[\![P]\!]_{\text{sync}}$ is an *asynchronous* π-calculus term that simulates synchronous communication; and Q is a *polyadic synchronous* π-calculus term, and $[\![Q]\!]_{\text{poly}}$ is a *monadic synchronous* π-calculus term that simulates polyadic communication.

$$
\begin{array}{lll}
[\![\overline{u}\langle v\rangle.P]\!]_{\text{sync}} & = (\nu\, c)(\overline{u}\langle c\rangle \mid c(y).(\overline{y}\langle v\rangle \mid [\![P]\!]_{\text{sync}})) & \text{where } y \notin \mathit{fv}(P), c \notin \mathit{fn}(P) \\
[\![u(x).P]\!]_{\text{sync}} & = u(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x).[\![P]\!]_{\text{sync}}) & \text{where } y \notin \mathit{fv}(P), d \notin \mathit{fn}(P) \\
[\![u(x_1,..,x_n).P]\!]_{\text{poly}} & = u(z).z(x_1)...z(x_n).[\![P]\!]_{\text{poly}} & \text{where } z \notin \mathit{fv}(P) \\
[\![\overline{u}\langle v_1,..,v_n\rangle.P]\!]_{\text{poly}} & = (\nu\, c)\overline{u}\langle c\rangle.\overline{c}\langle v_1\rangle...\overline{c}\langle v_n\rangle.[\![P]\!]_{\text{poly}} & \text{where } c \notin \mathit{fn}(P)
\end{array}
$$

To find an encoding of the **polyadic synchronous** π-calculus into the **monadic asynchronous** π-calculus, it would suffice to first encode polyadic synchronous terms into the monadic synchronous π-calculus, and then encode the synchronous term into the asynchronous π-calculus: $[\![[\![P]\!]_{\text{poly}}]\!]_{\text{sync}}$. We call this an **indirect encoding**. However, there is a **direct encoding** from the polyadic synchronous π-calculus into the monadic asynchronous π-calculus. The objectives of this exercise are:

(a) Find a direct encoding from the **polyadic synchronous** π-calculus into the **monadic asynchronous** π-calculus.

(b) Simulate the direct encoding in **Go** in a similar way to the example encodings given in the lectures.

(c) Instrument the Go code to compare the amount of messages and channels required by the direct and indirect encodings; i.e. count the number of channels and messages required by both encodings.

(d) Write a comment block at the beginning of the file explaining why the direct encoding requires less messages and channels than the indirect encoding.

Hint for the simulation: assume that you are sending integers, and set the polyadic encoding to to some fixed number (e.g. 3). Only simulate the encoding for sending/receiving integers. For communicating channels, you can use regular Go constructs.

Reminter:
Go is installed on the College machines, but is not available by default in the `PATH`. To access Go on the College machines, you have two options currently:

- the system installed version, Go 1.14.3, is at `/usr/lib/go-1.14/bin/go`;
- a more recent version, Go 1.15, has been compiled by ICT and is available at `/vol/linux/apps/go/bin/go`. This is the current release of Go as of September 2020.

To be able to use one of them without typing the full path to the binary, add the version of your choosing in the `PATH` with the `export` command as follows:

```
export PATH=$PATH:/path/to/go/directory
```
For example for the ICT version:
```
export PATH=$PATH:/vol/linux/apps/go/bin
```
The Go documentation is available at https://golang.org/doc/.

When preparing coursework at home, you may want to use a local version of Go instead of logging into College via SSH. If that is the case, documentation and binary distributions are available for the most common operating systems at https://golang.org/. This page also includes a link to the Go Playground, an online interpreter for Go that allows to test simple programs, at the cost of not showing any nondeterminism:https://play.golang.org/. We do *not* ask the handout to be playground-compatible, but it should run properly in a regular Go environment, e.g. the Go versions available on College machines.