# Exercices sur la logique temporelle (faits en classe le 27 mars)

Note: Les solutions indiquées plus bas ont été vérifiées à l'aide de l'outil `evaluator` de la boîte à outils CADP (nouvelle version). Quelques modifications mineures ont dû être effectuées:

- L'identificateur `in` est un mot-réservé en `LOTOS`. L'action `in` a donc du être renommée (`input`).
- Les modalités de la forme `[*-a1, a2]` ne sont plus acceptées par le vérificateur de logique temporelle. Je n'ai pas encore trouvé la nouvelle syntaxe qui serait équivalente.
- Dans le cas du type `boolean`, j'ai du utiliser la version ACT ONE de ce type.
- Les actions dans les opérateurs modaux `<>` et `[]` doivent être mis entre guillemets.

## 1. Processus `Pair`

Soit la définition suivante de processus:

```
PROCESS Pair[in, oui]: NOEXIT :=
  in; in; Pair[in, oui]
  []
  oui; Pair[in, oui]
ENDPROC
```

Formalisez les propriétés indiquées plus bas. Dans chaque cas, on suppose que la formule de logique temporelle s'exprime relativement à l'instance du processus `Pair[in, oui]` dans son ètat initial.

1. Dans son état initial, le processus permet l'action `oui`.

   ```
   <oui>true
   ```

2. Les seules actions que peut faire le processus sont `oui` et `in`.

   ```
   ALL( [*-in,oui]false and <in>true and (<oui>true or [in]<oui>true) )
   ```

3. À tout instant, il existe un futur tel que l'action `oui` deviendra possible.

   ```
   ALL( POT( <oui>true ) )
   ```

4. À tout moment, si on peut faire l'action `oui`, alors on pourra la faire à nouveau après avoir fait deux fois l'action `in`.

   ```
   ALL( [oui]false or [in][in]<oui>true )
   ```

   Ou encore:

   ```
   ALL( [oui][in][in]<oui>true )
   ```

## 2. Processus `Complementer`

Soit la définition suivante de processus:

```
PROCESS Complementer[in, out]: NOEXIT :=
  in  ?b: boolean;
  out !~b;
  Complementer[in, out]
ENDPROC
```

Formalisez les propriétés indiquées plus bas pour l'instance `Complementer[in, out]` de ce processus:

1. Le processus effectue une série sans fin d'actions `in` suivi immédiatement de `out`, ou vice-versa, et rien d'autre.

   ```
   ALL( ([in]<out>true or [out]<in>true) and [*-in,out]false )
   ```

2. Immédiatement après avoir reçue une valeur 0 sur le port `in`, la valeur 1 est transmise sur le port `out`, et symétriquement pour 1 et 0.

   ```
   ALL( [in !0]<out !1>true and [in !1]<out !0>true )
   ```