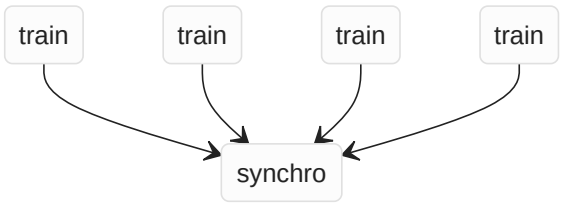
4. Problème de voie unique

Tronçon à voie unique, objectif : pas de de train en sens opposé.

Version simplifiée :

- capacité non bornée
- famine possible

Principe



Opération bloquante

- Envoyer un message sur un canal
  CSP: `canal!valeur` */ Go:* `canal<-valeur`
- Recevoir un message depuis un canal
  CSP: `canal?variable` */ Go:* `variable:=<-canal`

Alternative :

- action au choix
- ensemble de reception/émission

Interface : canaux ?

- `entrerEO`
- `entrerOE`
- `sortir`

Code d'un train

```
*[  entrerEO!_;
        ...
        sortie!_;
        ...
        entrerOE!_;
        ...
        sortie!_;
]
```

où `_` *est le message vide*

```
for {
        entrerEO <- _
        ...
        sortie <- _
        ...
        entrerOE <- _
```

```
        ...
        sortie <- _
}
```

Construction de l'activité de synchronisation : approche par conditions

- Énoncer les conditions d'acceptation par canal
  `entrerEO` : pas de train en sens OE
  `entrerOE` : pas de train en sens EO
  `sortie` : toujours faisable
- Variable d'état (interne à l'activité de synchro)
  `nbEO`, `nbOE`
- Invariant
  `(nbOE = 0) v (nbEO = 0)`

Activité de synchro

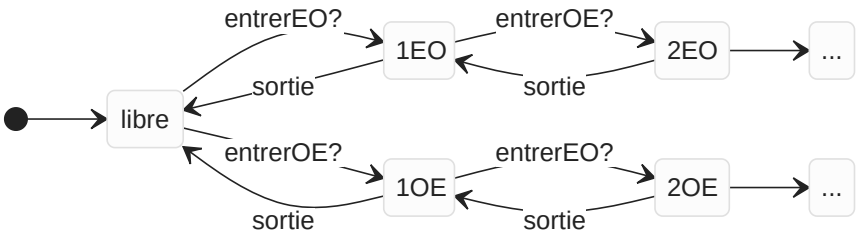
```
boucle
        alternative
                quels sont les canaux ouverts selon l'état courant
                selon la réception -> nouvel état
finboucle
```

```
*[
        nbOE = 0 -> entrerEO?_; nbEO++
        []
          nbEO = 0 -> entrerOE?_; nbOE++
        []
          sortir?_; nbEO > 0 -> nbEO--
                    [] _ -> nbOE--
]
```

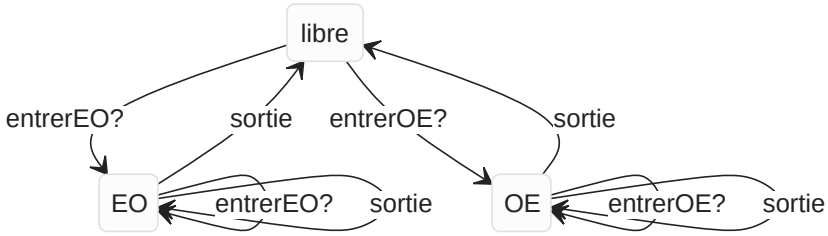
```
for {
        select {
                case <- when(nbEO = 0, entrerOE):
                        nbOE++
                case <- when(nbOE = 0, entrerEO):
                        nbEO++
                case <- sortie:
                        if nbEO > 0 {nbEO--} else {nbOE--}
        }
}
```

Approche par automate

L'état = l'ensemble de canaux ouverts (sur lesquels un message peut-être reçu)



Automate généralisé



Code

```
etat := libre
for {
        if etat == libre {
                select {
                        case <- entrerEO:
                                etat := EO
                                nb := 1
                        case <- entrerOE:
                                etat := OE
                                nb := 1
                }
        } else if etat == EO {
                select {
                        case <- entrerEO: //when(nb < N, entrerEO)
                                nb++
                        case <- sortir:
                                nb--
                                if nb == 0 {etat := libre}
                }
        } else {
                //etat == OE
                //sym de EO
        }
}
```

Capacité borné à N

- Approche condition
  `entrerEO : (nbOE = 0)^(nbEO < N)`
  `entrerOE : (nbEO = 0)^(nbOE < N)`
- Approche automate
  conditionner certaines ouverture

Famine

Un flux continu de trains dans un sens, tel qu'il y a toujours au moins un train sur la voie unique.
Empêche de manière permanente l'entrée des trains en sens opposé.
-> ne pas laisser entrer trop de trains successifs dans un sens s'il y a des demandes dans l'autre sens.

- soit savoir s'il y a des demandes d'écriture bloquées sur un canal => NON (pas fourni)
- soit enrichir l'interface

```
boucle
        preparer_entrerEO!_
        ...
        entrerEO!_
```

```
        ...
        sortie!_
        ...
finboucle
```

<- retour

#TD/SC