Ingeniería del Software 2

Taller 4 – Verificación de propiedades de Software

Ejercicio 1

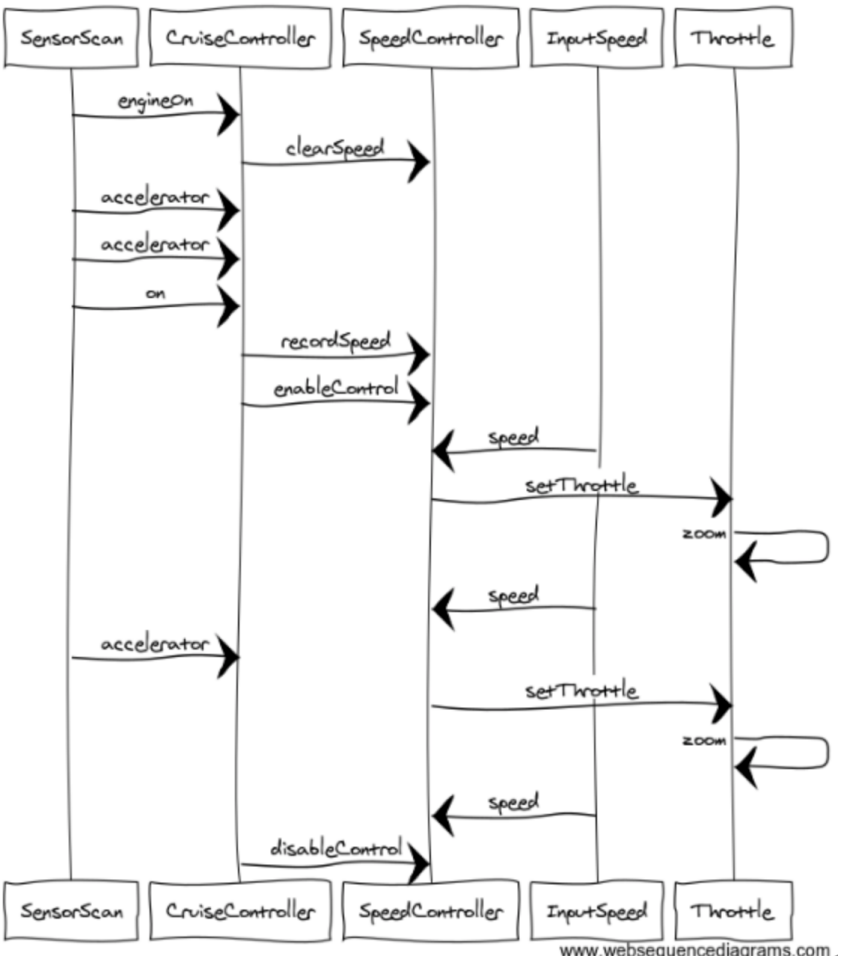
Considere el siguiente modelo del problema de los "filósofos comensales":

```
const N = 3

Philosopher = (think -> sit -> left.get -> right.get ->
               eat -> left.put -> right.put -> stand -> Philosopher).

Fork = (get -> put -> Fork).

||DP = (forall [i:0..N] (phil[i]:Philosopher ||
        {phil[i].left,phil[(i+1)%N].right}::Fork )).
```

a) ¿Cumple el modelo con ausencia de deadlock?
b) ¿Cumple el modelo con garantía de progreso para los eventos eat?

Ejercicio 2

Dado el FSP de un sistema de autocrucero (ver anexo) y el diagrama de secuencia que sigue, escribir un proceso FSP que compuesto con **CRUISECONTROLSYSTEM** produzca un deadlock si y sólo si el **CRUISECONTROLSYSTEM** NO puede exhibir el comportamiento descripto por el diagrama de secuencia. Es decir que al detectar deadlock en la composición se produce un ejemplo de traza descipta por el diagrama de secuencia, que **CRUISECONTROLSYSTEM** no puede hacer. Usando este método, informe si efectivamente el sistema satisface o no el diagrama de secuencia.



Ejercicio 3

En el modelo FSP de un controlador de un vehículo autónomo (ver más abajo) se describe un controlador que provee una capa funcional de más alto nivel que el vehículo mismo. Recibe comandos de un usuario y los resuelve ejecutando uno o mas comandos sobre el vehículo. Por ejemplo:

- Al recibir el comando de ir a una locación particular (*gotoLoc*), el controlador realiza movimientos (*moveN*, *moveS*, *moveW*, *moveE*) para llegar a la locación esperada e informa al usuario cuando llegó.
- Al recibir el comando de ascenso (*up*), el controlador asciende el vehículo siempre que no haya llegado a la altura máxima.
- De manera similar, el comando de descenso (*down*) es reenviado al vehículo si se respetó la altura mínima de vuelo.
- El comando de aterrizaje (*land*) es manejado por el controlador mediante una serie de comandos de descenso enviados al vehículo. El fin de la etapa de aterrizaje es notificada al usuario (*landed*). El despegue es manejado de manera similar con los eventos *takeOff* y *takenOff*.
- El usuario puede abortar tareas del controlador de larga duración como ser el despegue o aterrizaje (*abort*).

Para cada caso de uso descrito a continuacion, escribir un proceso FSP que compuesto con el sistema produzca un deadlock si y sólo si el sistema no satisface el caso de uso. Es decir que detectar deadlock produce un ejemplo de cómo el sistema viola el caso de uso.

a) Descripción: El usuario aborta el movimiento del vehículo antes de que este arribe a destino.
Precondición: El controlador recibió *gotoLoc* y aún no arribó a destino.
Acciones: El usuario aborta y luego aterriza.

b) Descripción: El usuario aborta el despegue del vehículo antes de que este termine de despegar.
Precondición: El controlador está despegando.
Acciones: El usuario aborta y luego aterriza.

Ejercicio 4

Escribir una propiedad en FSP que describa el siguiente proceso: entre el mensaje *goToLoc* y *abort* o *arrived* sólo puede haber *moveE, moveN, moveS, moveW.*

Ejercicio 5

En un modelo de ejecución Supervisor-Worker, múltiples procesos Worker están a la espera de tareas que un proceso Supervisor distribuye. Los procesos Worker toman una tarea, la resuelven, informan el resultado y luego quedan a la espera de otra tarea. El proceso Supervisor acepta una tarea y la entrega al primer Worker disponible o la retiene mientras todos los Workers están ocupados. Resuelva:

a) Modele un proceso **WORKER** que toma una tarea (*take*), inician su resolución (*start*), eventualmente terminan (*end*) y finalmente informan un resultado (*result*).
b) Modele un **SUPERVISOR** que acepta una tarea (*accept*) y la entrega a un **WORKER** disponible.
c) Indique si el sistema compuesto entre un **SUPERVISOR** y dos **WORKER** es libre de deadlocks.
d) Indique si siempre hay progreso en la resolución de tareas sobre el sistema del punto (c).
e) Modele un proceso para verificar si la distribución de carga en el sistema del punto (c) difiere a lo sumo uno.

1. FSP del Sistema de Autocrucero

```
set Sensors = {engineOn,engineOff,on,off,resume,brake,accelerator}
set Engine  = {engineOn,engineOff}
set Prompts = {clearSpeed,recordSpeed,enableControl,disableControl}

set Alphabet = {accelerator,brake,clearSpeed,disableControl,
         enableControl,engineOff,engineOn,off,on,
         recordSpeed,resume,setThrottle,speed,zoom}

SENSORSCAN = ({Sensors} -> SENSORSCAN).

INPUTSPEED = (engineOn -> CHECKSPEED),
CHECKSPEED = (speed -> CHECKSPEED |engineOff -> INPUTSPEED).

THROTTLE =(setThrottle -> zoom -> THROTTLE).

SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed}->DISABLED
            | enableControl -> ENABLED ),
ENABLED = ( speed -> setThrottle -> ENABLED
            |{recordSpeed,enableControl} -> ENABLED
            | disableControl -> DISABLED ).

set DisableActions = {off,brake,accelerator}

CRUISECONTROLLER = INACTIVE,
INACTIVE = (engineOn -> clearSpeed -> ACTIVE
            |DisableActions -> INACTIVE ),
ACTIVE   =(engineOff -> INACTIVE
            |on->recordSpeed->enableControl->CRUISING
            |DisableActions -> ACTIVE ),
CRUISING =(engineOff -> INACTIVE
            |DisableActions->disableControl->STANDBY
            |on->recordSpeed->enableControl->CRUISING ),
STANDBY  =(engineOff -> INACTIVE
            |resume -> enableControl -> CRUISING
            |on->recordSpeed->enableControl->CRUISING
            |DisableActions -> STANDBY ).

||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).

||CRUISECONTROLSYSTEM = (CONTROL||SENSORSCAN||INPUTSPEED||THROTTLE).
```

2. FSP de Vehículo Autónomo

```
set Alphabet =  {abort, arrived, down, goDown, goUp, up, gotoLoc, land, landed,
                 moveE, moveN, moveS, moveW, toggleVideo, tVideo, takeOff,
                 takenOff}
set UserActions = {gotoLoc, land, takeOff,  goDown, goUp, toggleVideo, abort}
set UserEvents = {arrived, takenOff, landed}
set VehicleActions = {down, up, moveE, moveN, moveS, moveW, tVideo}

CONTROLLER = IDLE,
IDLE = ( gotoLoc -> MOVING | gotoLoc -> arrived -> IDLE | land -> LANDING
           | takeOff -> TAKINGOFF | goUp -> up -> IDLE | goUp -> IDLE
           | goDown -> down -> IDLE | goDown -> IDLE
           | toggleVideo -> tVideo -> IDLE),
MOVING = ({moveN, moveS, moveE, moveW} -> MOVING | abort -> IDLE
           | {moveN, moveS, moveE, moveW} -> arrived -> IDLE),
LANDING = (down -> LANDING | down -> landed -> IDLE),
TAKINGOFF = (up -> TAKINGOFF | up -> (takenOff -> IDLE
           | abort -> IDLE) | abort -> IDLE).

USER = ({UserActions,UserEvents} -> USER).
VEHICLE = ({VehicleActions} -> VEHICLE).

||SYSTEM = (USER || CONTROLLER || VEHICLE).


//Ayudita:

||Check_UseCaseMoving = (SYSTEM || USECASE_Moving).

||Check_UseCaseTakingOff = (SYSTEM || USECASE_TakingOff).
```