Ingeniería del Software II
Evaluación Parcial

Ejercicio 1. Una torre de control en un aeropuerto debe administrar el acceso a una única pista. Un avión en un hangar puede pedir permiso de despegue, cuando le es concedido comienza el taxing y luego despega. Luego de un tiempo de vuelo el avión puede pedir permiso de aterrizaje. La torre puede concederle el permiso o requerirle hacer maniobras de holding (dejándolo en espera). Cuando la torre concede el permiso el avión comienza el descenso y aterriza. Si en cualquier momento dos aviones intentan acceder a la pista simultáneamente (ya sea para despegar o aterrizar) ocurre una colisión.

Modele usando FSP:

a. Un proceso Avión, que comienza en un hangar, puede pedir *permiso de despegue*, hacer el *taxing* y el *despegue*. Luego de su *vuelo* puede pedir *permiso de aterrizaje*, hacer maniobras de *holding*, *descender* y *aterrizar*.
b. Un proceso Torre, que controle el tráfico aéreo respondiendo a los pedidos de los aviones y que garantice que en ningún momento haya más de un avión utilizando la pista. Considere, además, que la torre debe priorizar los pedidos de aterrizaje por sobre los de despegue.
c. La composición paralela entre tres procesos Avión y un proceso Torre indicando claramente prefijos, renombres y priorización de eventos.
d. Una propiedad que indique que todos los aviones van a poder volar infinitas veces.
e. Una propiedad que indique que no puede ocurrir una colisión.

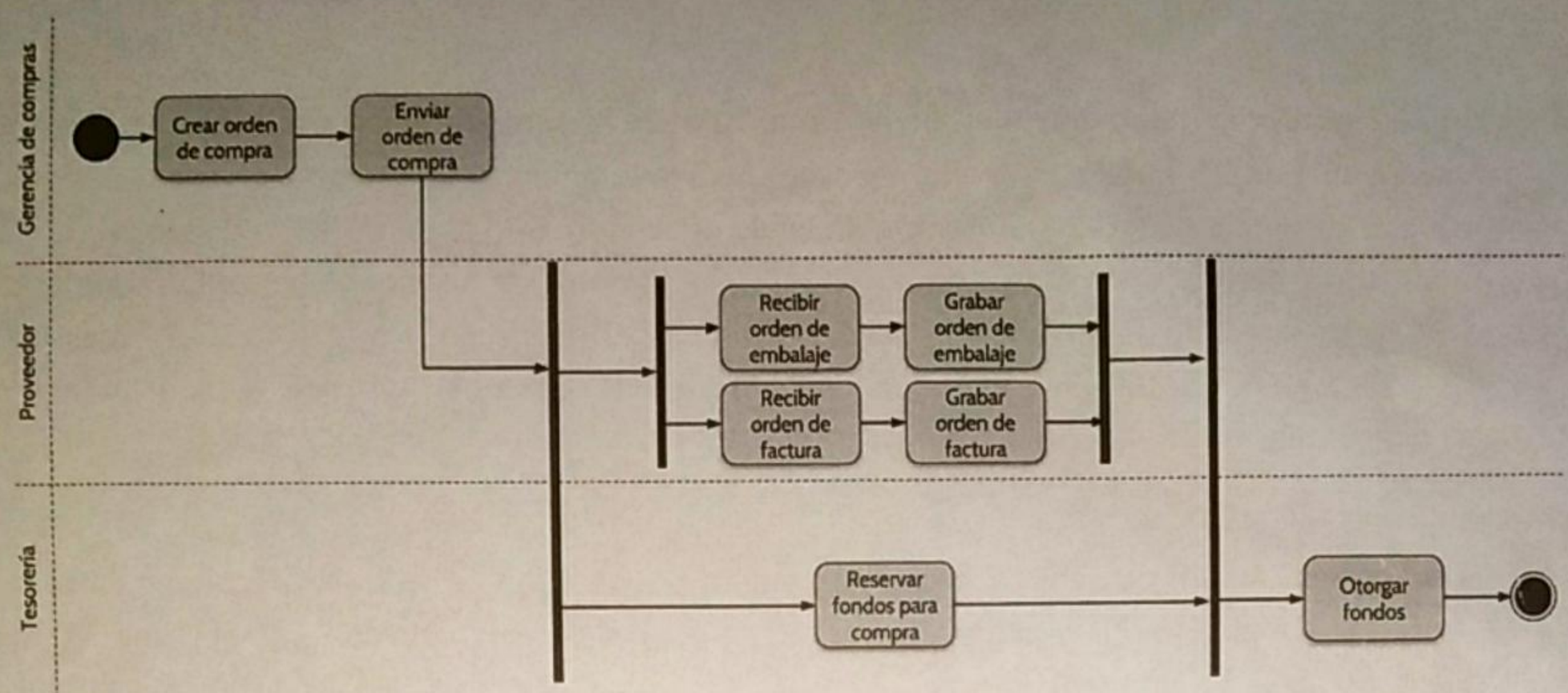
Ejercicio 2. Indique cuáles de los siguientes procesos descritos en FSP son bisimilares o débilmente bisimilares entre sí. Justifique mostrando una relación o un contraejemplo. Luego indique cuáles son minimales.

```
ENV = (procedure-> ENV | stopPump -> STOPED),
  STOPED = (cleanUp -> MAINTANANCE | refill -> STARTED),
  MAINTANANCE = (endProcedure -> STOPED),
  STARTED = (startPump -> ENV) \ {refill,cleanUp}.

ENV2 = (procedure-> PROCEED | stopPump -> STOPED),
  PROCEED = (procedure-> ENV2 | stopPump -> STOPED),
  STOPED = (endProcedure -> STOPING | startPump -> ENV2),
  STOPING = (endProcedure -> STOPED | startPump -> ENV2).

ENV3 = (procedure-> ENV3 | stopPump -> STOPING),
  STOPING = (endProcedure -> STOPING | startPump -> ENV3).
```

Ejercicio 3. Dado el siguiente diagrama de actividad:



Escriba:

a. Un proceso FSP que al componerlo produzca un deadlock si el comportamiento descrito por el diagrama no es posible.
b. Una propiedad que detecte si *recibirOrdenEmbalaje* y *recibirOrdenFactura* pueden suceder concurrentemente con su respectiva actividad de grabado.
c. Una propiedad que determine si el evento *otorgarFondos* puede suceder infinita veces.

Ejercicio 4. Dado el siguiente Kripke (descrito con un modelo NuSMV):

```
MODULE Player
  VAR play : {rock,paper,scissors};

MODULE Main
  VAR p1 : Player;
      p2 : Player;
  DEFINE result := case
                     p1.play=p2.play : 0;
                     p1.play=rock & p2.play=scissors : 1;
                     p1.play=rock & p2.play=paper : -1;
                     p1.play=scissors & p2.play=paper : 1;
                     p1.play=scissors & p2.play=rock : -1;
                     p1.play=paper & p2.play=rock : 1;
                     p1.play=paper & p2.play=scissors : -1;
                   esac;
```

a. Transformar el Kripke (representado por el modelo Main) a un autómata de Büchi.
b. Dar la fórmula LTL que represente: "Siempre que el jugador 1 elige piedra gana (en ese turno)" y construya el autómata de Büchi necesario para verificar la fórmula.
c. Componga los dos Büchi e indique si el modelo satisface la fórmula del punto b.