Materia: Ingeniería de Software 2
Tema: Programas Concurrentes
Práctica: 01 - Modelado de Procesos Secuenciales y Concurrentes

---

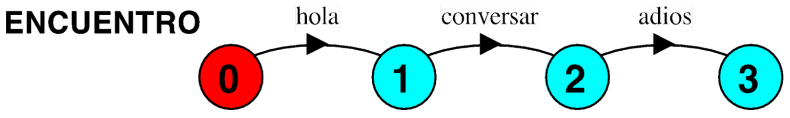
La herramienta MTSA se encuentra en las máquinas del laboratorio en: /opt/mtsa/

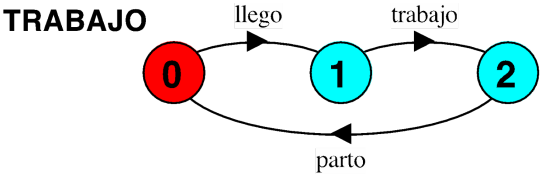
También se puede bajar de: http://mtsa.dc.uba.ar/download/MTSA_latest.zip

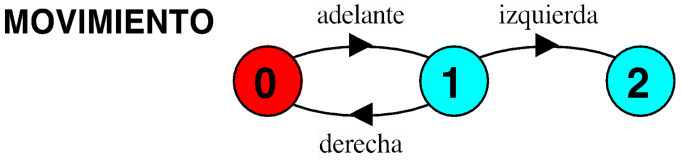
La sintaxis de FSP puede encontrarse en: https://www.doc.ic.ac.uk/~jnm/LTSdocumention/FSP-Syntax.html

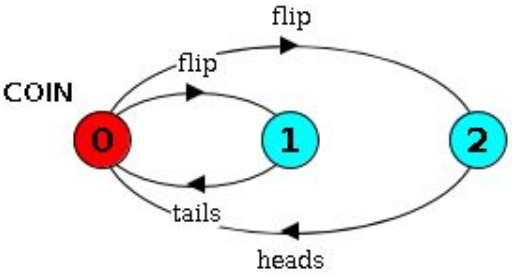
---

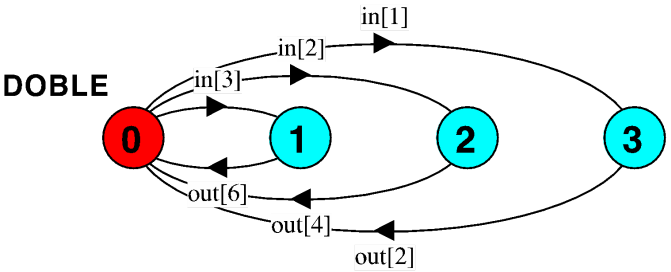
Ejercicio 1. Para cada uno de los siguientes LTS, dar una expresión en FSP (lo más compacta posible) que lo tiene como semántica.

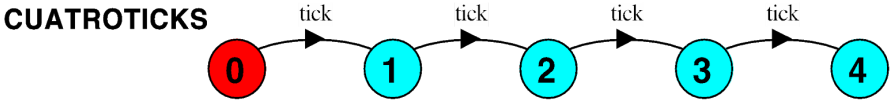












Ejercicio 2. Un circuito digital recibe una secuencia de señales *trigger* y da como output 0 y 1 de manera alternada. Modelar el circuito usando *FSP*, y verificar usando MTSA que puede producir la traza:

trigger → 1 → trigger → 0 → trigger → 1 → trigger → 0 → …

Ejercicio 3. Modele el controlador de un horno microondas. El horno sólo puede ponerse en marcha (estado ON) sólo si la puerta está cerrada. Si se abre la puerta estando el horno prendido, el horno debe apagarse automáticamente (estado OFF). Cuando el horno no está en funcionamiento puede seleccionarse un modo de cocción (estado SELECTED[mode:Mode]). Los modos de cocción posibles son descongelar (0 por default), calentar (1) y grill (2). Luego de un tiempo de comenzado, el proceso de cocción termina emitiendo un beep.

Considere los siguientes eventos:

- *open*, que indica que la puerta fue abierta;
- *close*, que indica que la puerta fue cerrada;
- *start*, que indica que el usuario inició la cocción;
- *beep*, que indica que la cocción terminó; y
- *defrost*, *heat*, *grill*, que indican la selección del modo de cocción.

Ejercicio 4. Una variable guarda valores entre 0..*N* y permite ser leída y escrita a través de eventos *read* y *write*. Modelar la variable como un proceso, `VARIABLE`, usando *FSP*. Considere que la variable empieza inicializada en 0.

Para N=2, una traza que el modelo es resultante debería poder exhibir es:

write[2] → read[2] → read[2] → write[1] → write[0] → read[0] → ...

Para N=2, una traza que el modelo es resultante NO debería poder exhibir es:

write[2] → read[2] → read[1] → …

Ya que en esta traza se lee un valor distinto al último escrito.

Luego modele (a) un proceso `ESCRITOR` que escribe un valor entre 0 y N; (b) un proceso `LECTOR` que lee la variable y si su valor es distinto de cero lo imprime mediante el evento imprimir[i]; y (c) calcule la composición paralela de los tres procesos anteriores.

Ejercicio 5. Un museo que puede albergar hasta **N** turistas, permite entrar a los turistas por la entrada oriental y salir por la salida occidental. Arribos y partidas se señalan al controlador del museo mediante señales *entry* y *exit* emitidas por molinetes. Cuando el museo debe abrir, el director del mismo le da la señal *open* al controlador y el controlador permite el ingreso y egreso de visitantes. A la hora de cerrar, el director da la señal *close* al controlador que a partir de ese momento solo permite egresos. Cuando el museo está vacío, el director recibe la señal *empty* del controlador. El director no reabre el museo hasta que el museo se encuentra sin turistas dentro.

El alfabeto para el museo es *{entry, exit, open, close, empty}*. Modele en FSP los procesos que componen el museo (y calcule su composición):

a) `ESTE`: que representa el molinete en la entrada este;
b) `OESTE`: que representa el molinete en la entrada oeste;
c) `DIRECTOR`: que representa el director del museo; y
d) `CONTROL`: que codifica la lógica de control deseada.

Ejercicio 6. El controlador para una montaña rusa solo permite que el tren parta cuando está lleno. Pasajeros que llegan a la plataforma se registran con el controlador mediante un molinete. El controlador le da la señal al tren para que parta cuando hay suficientes personas en la plataforma como para llenar el tren que tiene capacidad M. El tren entonces da la vuelta a la montaña rusa y luego espera por otros M pasajeros. Un máximo de M pasajeros puede ocupar la plataforma. Ignorar los detalles de sincronización relacionados con la suba de pasajeros al tren desde la plataforma y salida del coche. La montaña rusa tiene entonces tres procesos: `MOLINETE`, `CONTROL` y `TREN`. `MOLINETE` y `CONTROL` interactúan a través de la acción *pasajero* que indica una llegada de un pasajero a la plataforma. `CONTROL` y `CAR` interactúan vía la acción *partida* que indica que el tren partió. Proveer un diagrama de estructura y una descripción FSP para cada proceso y la composición.

Ejercicio 7. Extender el modelo `CLIENT_SERVER` de las teóricas para que:

a) El servidor pueda ser usado por dos clientes.

b) Además del punto (a), un llamado de un cliente pueda terminar con un `timeout` en vez de una respuesta del servidor. ¿Qué pasa con el servidor en este caso?

c) Además de los puntos (a) y (b), el servidor pueda hacer `timeout` si el cliente no responde. ¿El problema se soluciona?