Universidad de Buenos Aires
Facultad De Ciencias Exactas y Naturales
Departamento de Computación
Año 2017 - 2do Cuatrimestre

Simulación de Eventos Discretos

TRABAJO PRÁCTICO Nº <2>
TEMA:<Manejo del Inventario de una Industria>
FECHA:<27 de noviembre de 2017>

INTEGRANTES:

CERVETTO, Marcos - #FIUBA
<cervettomarcos@gmail.com>

MARCHI, Edgardo - #FIUBA
<edgardo.marchi@gmail.com>

PECKER MARCOSIG, Ezequiel - #FIUBA
<ezepecker@gmail.com>

Índice

1. Objetivo y Enunciado

2. Modelo Conceptual

2.1. Motivación

Una compañía que vende un único producto está interesada en estudiar la forma óptima de ordenamiento de las unidades producidas en el almacén.

En la Figura 1 se muestra un esquema de la industria, donde se observa la ubicación del inventario.

Para este trabajo, el bloque Inventario se va a modelizar con un autómata celular.

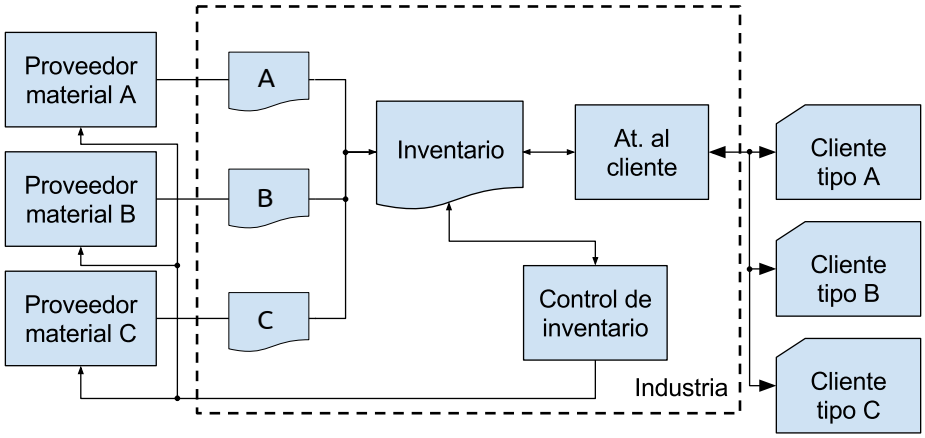


Figura 1: Esquema del problema planteado.

2.2. Autómata Celular

El bloque atómico DEVS correspondiente al Inventario que se indica en la Figura 1 va a ser reemplazado por tres bloques cell-DEVS, según la Figura 2.

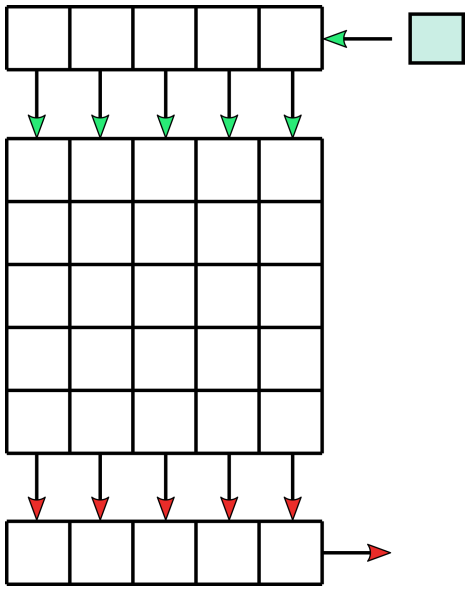


Figura 2: Inventario con cell-DEVS.

El primer bloque corresponde a una fila de celdas que manejarán la ubicación inicial de los productos dentro del inventario. Del trabajo práctico 1 se puede recordar que cada producto tiene una fecha de vencimiento asociada. Esta fecha será la que determine la columna en la que se apilará a cada producto. Por ejemplo: si falta menos de una semana para su vencimiento se apilará en la columna 0 (más a la derecha), si falta entre 1 y 2 semanas en la columna 1, entre 2 y 3 en la columna 2, etc. Este proceso se esquematiza en la Figura 3.

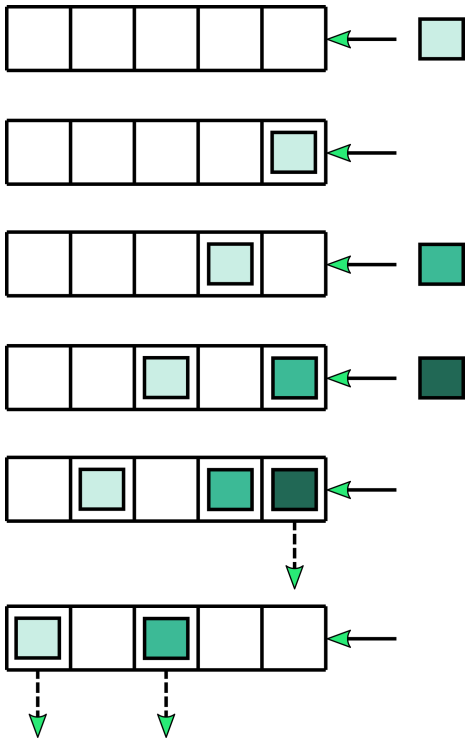


Figura 3: Ingreso de productos al inventario.

El segundo bloque será el inventario propiamente dicho. Es una grilla donde las columnas representan posiciones de apilamiento de productos. Las entradas de productos se realizan por la parte superior de cada columna, de forma de ir apilándolos. La salida de productos se realiza por la parte inferior de cada columna. Periódicamente se chequea la fecha de vencimiento de cada producto, y si cumple la condición de la columna siguiente a la derecha, por ejemplo que falte menos de N semanas para que perezca, el producto se intenta mover a esa columna.

La salida de productos está físicamente cercana a la columna derecha (la que contiene a los productos más próximos a vencer). El encargado de retirar productos demora un tiempo hasta llegar a la columna N por lo que idealmente se prefiere retirar los productos de la columna 0. Sin embargo si no hay productos con una fecha de vencimiento tan próxima, deberá recorrer las columnas hasta llegar a un producto. Este proceso de retiro de elementos se modela con el tercer bloque cell-DEVS, en este caso también de una sola fila. Todo este proceso se puede observar en la Figura 5. En ésta el círculo rojo corresponde a una demanda de un producto. La fecha de vencimiento se grafica según el esquema de colores de la Figura 4, donde los colores más oscuros corresponden a productos más próximos al vencimiento.

Figura 4: Codificación de colores por fecha de vencimiento.

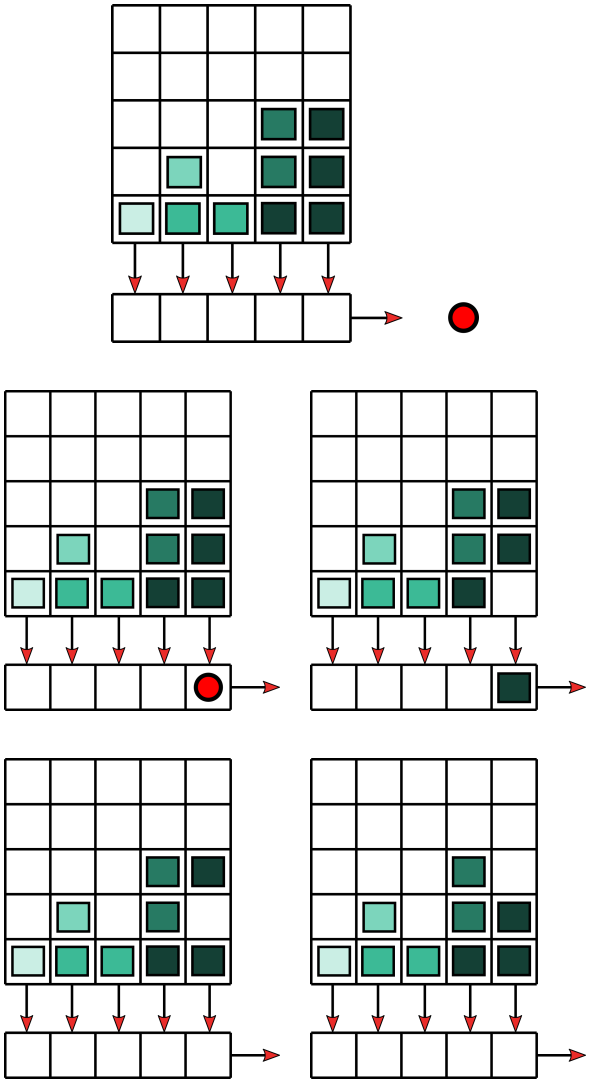


Figura 5: Autómata celular del inventario.

2.3. Celdas del inventario

El objetivo del autómata celular es ordenar los productos en el almacén de modo que aquellos con fecha de vencimiento más próxima queden ubicados espacialmente más cerca de la salida, de modo de ser despachados más rápido. De esta manera se busca reducir la cantidad de productos vencidos dentro del almacén. Para esto, periódicamente se revisan las fechas de vencimiento de los productos estampadas en un código de barras en cada unidad. Su ubicación depende del valor de su fecha de vencimiento. Los productos sólo pueden ser movidos si la columna adyacente a la derecha tiene una ubicación disponible a la altura del producto. Cabe destacar que si la ubicación inferior a donde se encuentra un producto está libre, el producto baja hasta estar apoyado sobre otro producto o sobre el suelo del depósito. Por estos motivos el vecindario propuesto es el de Moore y se puede ver en la Figura 6.

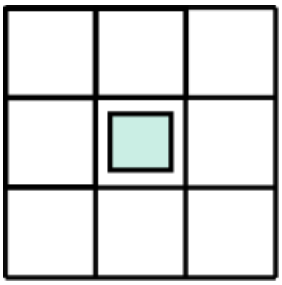


Figura 6: Vecindario.

Las preguntas a responder mediante simulaciones son:

- *¿Qué política de ordenamiento permite reducir la cantidad de unidades vencidas al momento del despacho?*
- Y conectada con la pregunta anterior, *¿qué política de ordenamiento permite disminuir el tiempo necesario para despachar una unidad?*

3. Descripción Formal

3.1. *Top-Model*

El diagrama de bloques correspondiente a cada uno de los autómatas celulares descriptos en la especificación funcional se puede ver en la Figura 7.

3.2. Cinta Transportadora

La cinta transportadora es el dispositivo por el que ingresan los productos al inventario. La cinta tiene su entrada `in` por la derecha en la celda $[0, N]$ con $N = 5$ en este caso. Además cada celda tiene asociadas una entrada `ini` y una salida `outi`, como se observa en la Figura 8.

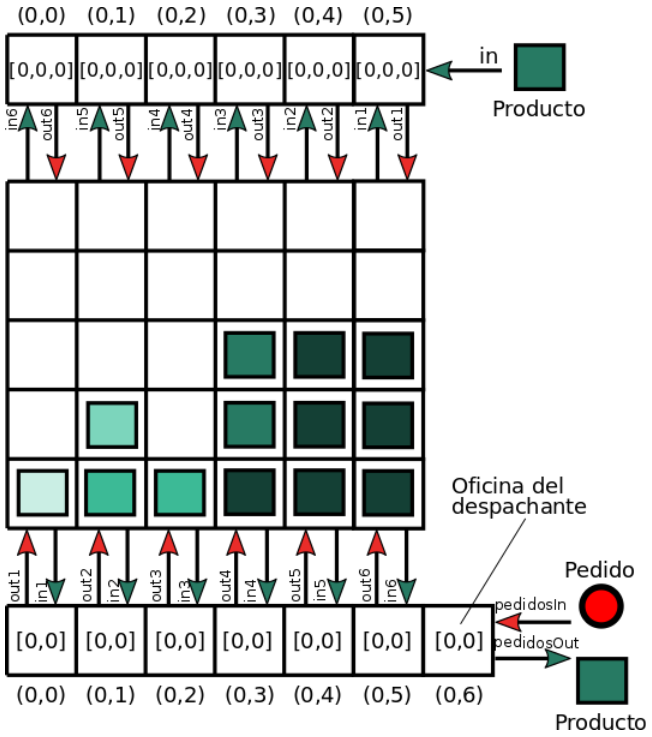


Figura 7: Interconexión de autómatas celulares en el `top model`.

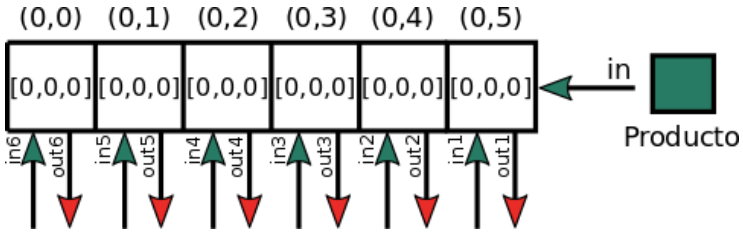


Figura 8: Autómata celular de la cinta transportadora.

Los productos ingresan por la entrada `in` y se van desplazando hacia la izquierda a una velocidad representada por el tiempo entre ejecuciones de las reglas (100 ms).

Para cada celda en este autómata celular importa solamente el valor de la celda anterior y el de la celda posterior. Por este motivo el vecindario está definido en la Figura 9.

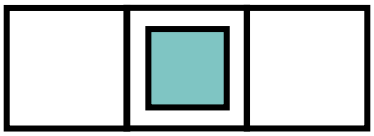


Figura 9: Vecindario de la cinta transportadora.

Cada posición de la cinta transportadora está asociado a una columna en el inventario. A su vez, cada columna en el inventario tiene asignado un rango de fechas de vencimiento posibles, estando más hacia la izquierda las columnas asociadas a vencimientos más remotos.

Las consultas de lugar disponible en una columna del inventario se realizan enviando un valor de fecha de vencimiento imposible, en este caso igual a -1. El inventario debe responder con 0 si hay lugar en la columna por la que se consultó o un número no nulo en caso contrario. En caso que la respuesta del inventario sea que no hay lugar en la columna por la que se consultó entonces la cinta debe hacer avanzar al producto hacia la izquierda una posición más y volver a preguntar por la disponibilidad de espacio en el inventario.

Cada elemento en la cinta transportadora se representa con una tupla de tres elementos: `[ID columna, producto, indicador]`. El `ID columna` representa el rango de fechas de vencimiento asociados a la celda y a la columna respectiva del inventario. El elemento `producto` es el producto en sí y está representado por su fecha de vencimiento. Por último, el elemento `indicador` puede tener distintos significados dependiendo de su valor y los valores que puede tomar son: 0, 1 ó 2. Si vale 1 significa que el producto estaba listo para ser ubicado en la columna del inventario asociada a la celda actual de la cinta transportadora pero se recibió un mensaje indicando que no había lugar, si en cambio vale 2 significa que ninguna de las columnas del inventario por las que se consultó tenía espacio y entonces ese producto permanecerá en la cinta transportadora.

Los elementos de las tuplas se acceden utilizando el signo de exclamación e indicando a continuación el elemento que se quiere acceder. Así por ejemplo el elemento 0 o `ID columna` de la celda actual se accede mediante $(0, 0)!0$.

Las reglas tienen la siguiente sintaxis:

```
rule : { resultado } { demora } { condicion }
```

Las reglas asociadas a las transiciones locales en las celdas del autómata y que representan a la descripción anterior se detallan a continuación:

```
[cinta-reglas]

rule : { [(0,0)!0,(0,1)!1,0] } { 100 } { NOT isUndefined((0,1)
    ↪ !1) AND (0,1)!1 != 0 AND (0,1)!1 > (0,1)!0 + time/1000
    ↪ }

rule : { [(0,0)!0,0,0] } { 100 } { NOT isUndefined((0,-1)!1)
    ↪ AND (0,0)!1 != 0 AND (0,0)!1 > (0,0)!0 + time/1000 AND
    ↪ (0,-1)!1 = 0 }

rule : { [(0,0)!0,(0,1)!1,0] } { 100 } { (0,1)!2 = 1 }

rule : { [(0,0)!0,0,0] } { 100 } { (0,0)!2 = 1 AND NOT
    ↪ isUndefined((0,-1)!1) AND (0,-1)!1 = 0 }

rule : { [(0,0)!0,(0,0)!1,2] } { 100 } { (0,0)!2 = 1 AND (
    ↪ isUndefined((0,-1)!1) OR (0,-1)!1 != 0) }

rule : { [(0,0)!0,(0,0)!1+send(output,-1),0] } { 100 } { (0,0)
    ↪ !2 != 2 AND (0,0)!1 != 0 AND (0,0)!1 <= (0,0)!0 + time
    ↪ /1000 }

rule : { (0,0) } 0 { t }
```

1. Las primeras dos reglas están relacionadas con el movimiento de un producto hacia la izquierda. Si una celda está vacía y la celda que está a su derecha tiene un producto que de acuerdo con su fecha de vencimiento debe moverse hacia la izquierda entonces el producto se mueve de la celda de la derecha a la celda actual.

2. Las siguientes dos reglas están relacionadas también con el movimiento de un producto hacia la izquierda. Si una celda está vacía y la celda que está a su derecha tiene un producto cuyo elemento `indicador` de la tupla es 1 significa que la cinta transportadora consultó al inventario para mover ese producto en el paso anterior y éste le respondió que no había lugar. Entonces lo que se hace es mover ese producto a la siguiente celda hacia la izquierda.

3. La quinta regla se ejecuta cuando el producto no puede ser descargado en el inventario porque no hay lugar en las columnas que pregunta y alcanza el extremo izquierdo de la cinta transportadora. Esta condición se indica escribiendo un 2 en el elemento `indicador` del producto en la celda.

4. La sexta regla está relacionada con la consulta al inventario por lugar para alojar un producto en la columna correspondiente. Si una celda tiene un producto cuya fecha de vencimiento está dentro del rango de fechas asignado a dicha celda entonces el producto se debe pasar al inventario a la columna correspondiente a dicha celda. Para esto, la cinta consulta

por la disponibilidad de espacio en la columna enviando −1 por la salida asociada con la celda `outi`. La respuesta a esta consulta se recibe por la entrada `ini`. Si el valor devuelto es 0 significa que hay espacio y entonces se puede descargar el producto (ver reglas asociadas con la transición en el puerto de entrada `[inventario regla]`). Se puede ver que si el elemento `indicador` de la celda actual es 2 entonces esta regla no se ejecuta, para de esta forma evitar la repetición de consultas al inventario.

5. Finalmente, la última regla es la regla por omisión que es siempre verdadera. Siempre debe existir al menos una regla verdadera.

Para terminar de definir el modelo, se deben especificar las reglas de comportamiento ante transiciones de los diferentes puertos de entrada se definieron reglas que se ejecutan ante la aparición de las distintas entradas. Así, ante el arribo de un producto por la entrada `in` se ejecuta la siguiente regla:

```
[in-regla]
rule : { [(0,0)!0,portValue(thisPort),0] } { 1 } { t }
```

Se observa que:

1. Lo que se hace es copiar el producto que acaba de llegar en la celda conectada a la entrada `in`.

Finalmente, como se mencionó más arriba, cuando el producto llega a la celda que le corresponde en la cinta transportadora se genera una consulta al inventario para saber si existe lugar en la columna correspondiente. Las reglas que se evalúan cuando llega la respuesta son:

```
[inventario-regla]
rule : { [(0,0)!0,0+send(output,(0,0)!1),0] } { 1 } {
    ↪ portValue(thisPort)=0 }
rule : { [(0,0)!0,(0,0)!1,1] } { 1 } { portValue(thisPort)!=0
    ↪ }
```

Se puede ver que:

1. Como se utiliza el comando `thisPort` esta regla es válida para todas las celdas de la cinta transportadora.

2. Si la respuesta que recibe la celda de parte del inventario es nula entonces hay lugar y se envía el producto por el mismo puerto por el que se realizó la consulta: `[(0,0)!0,0+send(output,(0,0)!1),0]` y se coloca un 0 en su lugar.

3. Si en cambio no hay lugar entonces se indica colocando un 1 en el elemento `indicador` de la tupla: `[(0,0)!0,(0,0)!1,1]`.

Este conjunto de reglas junto con la estructura graficada en la Figura 8 definen el modelo de la cinta transportadora. Los links y la asignación de reglas se pueden ver en la siguiente definición dentro del archivo `cinta.ma`.

```
[cinta]
type : cell
dim : (1,6)
delay : transport
defaultDelayTime  : 0
border : nowrapped
neighbors : cinta(0,-1) cinta(0,0) cinta(0,1)
initialvalue : 0
initialCellsValue : cinta.val
in : in in1 in2 in3 in4 in5 in6
out : out1 out2 out3 out4 out5 out6
link : in in@cinta(0,5)
link : in1 in1@cinta(0,5)
link : in2 in2@cinta(0,4)
link : in3 in3@cinta(0,3)
link : in4 in4@cinta(0,2)
link : in5 in5@cinta(0,1)
link : in6 in6@cinta(0,0)
link : output@cinta(0,5) out1
link : output@cinta(0,4) out2
link : output@cinta(0,3) out3
link : output@cinta(0,2) out4
link : output@cinta(0,1) out5
link : output@cinta(0,0) out6
portintransition : in@cinta(0,5) in-regla
portintransition : in1@cinta(0,5) inventario-regla
portintransition : in2@cinta(0,4) inventario-regla
portintransition : in3@cinta(0,3) inventario-regla
portintransition : in4@cinta(0,2) inventario-regla
portintransition : in5@cinta(0,1) inventario-regla
portintransition : in6@cinta(0,0) inventario-regla
localtransition : cinta-reglas
```

3.3. Inventario

Según lo descripto en la sección 2.2 el inventario de productos es el encargado de almacenarlos. Su modelo estructural se muestra en la Figura 10.

Este autómata celular se puede ver como una estantería en donde los productos se acomodan en la celda que les corresponde de acuerdo a las reglas que se explican más adelante. Los productos ingresan al inventario provenientes de la cinta transportadora conectada a los puertos de entrada asociados a las celdas superiores. Los productos egresan del inventario por los puertos de salida asociados a las celdas en la fila inferior.

Para cada celda en este autómata celular interesa el valor de las 8 celdas alrededor. Por este motivo el vecindario está definido en la Figura 11. El ordenamiento de productos está basado en la fecha de vencimiento de los mismos. Cada columna tiene asociada una fecha relativa de vencimiento, por ejemplo, hay columnas que pueden contener los productos que les falte 1 mes para ven-

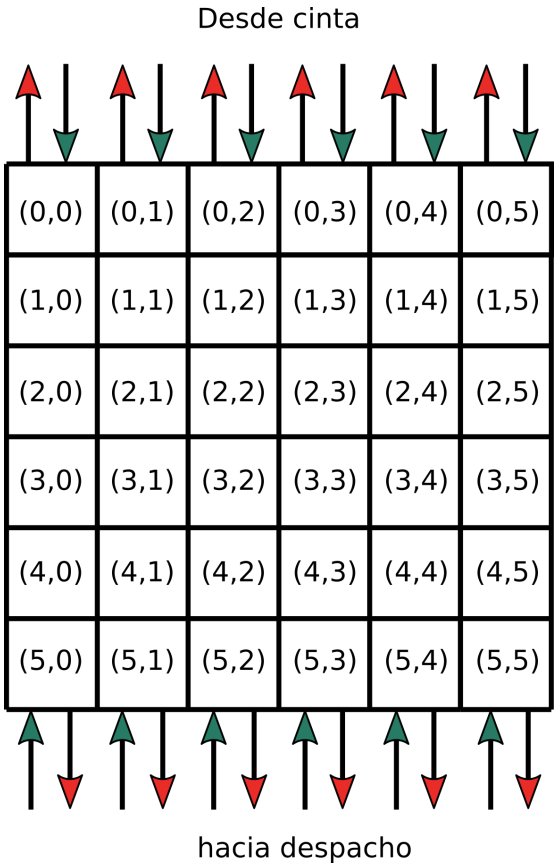


Figura 10: Modelo del inventario de productos.

cer, otra para los que les falte 2 meses para vencer, otra de 3 meses, etc. Las columnas con tiempo de vencimiento mayor son las de la izquierda de la grilla. Esto es así, porque la salida más próxima de productos en la zona de despachos se realiza en la columna de la derecha.

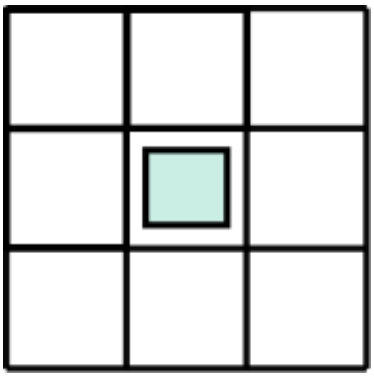


Figura 11: Vecindario de Moore para el inventario de productos.

El acomodamiento de los productos dentro del inventario es el siguiente: los productos van formando una pila en la columna donde entran y sólo se pueden mover hacia abajo o hacia la derecha. Un producto se podrá mover a la derecha sólo si el paso del tiempo hace que el vencimiento del mismo ya pueda pertenecer a la columna de la derecha. Si la posición se encuentra ocupada, no hay movimiento. Puede ocurrir una situación de disputa si se libera una posición que puede se tomada tanto por el bloque superior como por el bloque contiguo. Esta situación se representa en la Figura 12. En este caso, se comparan las fechas de vencimiento de ambos productos y la ocupa el de vencimiento más próximo.

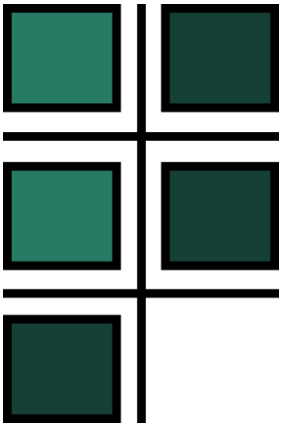


Figura 12: Disputa entre productos por un lugar recién liberado.

Cada celda del inventario se representa por una tupla de 3 elementos: `[rango, producto,flag]`. El elemento `rango` de la tupla identifica a la columna del inventario a la que pertenece la celda y representa el rango *relativo* de fechas de vencimiento de productos que se pueden alojar en la columna. Por ejemplo, en la columna i se puede alojar aquellos productos cuyas fechas de vencimiento

estén comprendidas entre 1 mes y 2 meses. Por otro lado, el elemento `producto` de la tupla es la fecha de vencimiento *absoluta* del producto alojado en dicha celda. Por ejemplo, la fecha de vencimiento del producto en la celda es 27 de noviembre de 2017. Por último, el elemento `flag` de la tupla se utiliza para generar eventos en las celdas y de esta manera lograr que un producto avance de acuerdo con el paso del tiempo.

Las reglas asociadas a las transiciones locales en las celdas del autómata se detallan a continuación:

```
[inventario-reglas]

%Hacia abajo si esta vacio y el de la izquierda quiere entrar tb
%Tiene prioridad el de arriba en caso de empate

%Gana el de arriba
rule : { [(0,0)!0,(-1,0)!1,0] } 100 {not isUndefined((-1,0)!1) and not isUndefined((0,-1)!1) and (-1,0)!1!=0 and (0,-1)!1!=0 and (0,-1)!1<(0,-1)!0+time/1000 and (0,0)!1=0 and (-1,0)!1<=(0,-1)!1}
rule : { [(0,0)!0,0,0] }          100 {not isUndefined((1,-1)!1) and not isUndefined((1,0)!1)  and (1,0)!1=0   and (1,-1)!1!=0 and (1,-1)!1<(1,-1)!0+time/1000 and (0,0)!1!=0 and (0,0)!1<=(1,-1)!1}

%caso particular: gana el de arriba porque no hay nada a la izquierda
rule : { [(0,0)!0,(-1,0)!1,0] } 100 {not isUndefined((-1,0)!1) and not isUndefined((0,-1)!1) and (-1,0)!1!=0 and (0,-1)!1=0 and (0,0)!1=0 }
rule : { [(0,0)!0,0,0] }          100 {not isUndefined((1,-1)!1) and not isUndefined((1,0)!1)  and (1,0)!1=0   and (1,-1)!1=0 and (0,0)!1!=0 }

%caso particular: gana el de arriba porque el de la izquierda no puede pasar
rule : { [(0,0)!0,(-1,0)!1,0] } 100 {not isUndefined((-1,0)!1) and not isUndefined((0,-1)!1) and (-1,0)!1!=0 and (0,-1)!1!=0 and (0,-1)!1>=(0,-1)!0+time/1000 and (0,0)!1=0}
rule : { [(0,0)!0,0,0] }          100 {not isUndefined((1,-1)!1) and not isUndefined((1,0)!1)  and (1,0)!1=0   and (1,-1)!1!=0 and (1,-1)!1>=(1,-1)!0+time/1000 and (0,0)!1!=0}

%caso particular: columna de la izquierda, solo se puede bajar
rule : { [(0,0)!0,(-1,0)!1,0] } 100 {not isUndefined((-1,0)!1) and isUndefined((0,-1)!1) and (-1,0)!1!=0 and (0,0)!1=0}
rule : { [(0,0)!0,0,0] }          100 {isUndefined((1,-1)!1) and not isUndefined((1,0)!1)  and (1,0)!1=0  and (0,0)!1!=0}
```

```
%Gana el de la izquierda
rule : { [(0,0)!0,(0,-1)!1,0] } 100 {not isUndefined((-1,0)!1)
    ↪  and not isUndefined((0,-1)!1) and (0,0)!1=0 and (0,-1)
    ↪ !1!=0 and (0,-1)!1<(0,-1)!0+time/1000 and (-1,0)!1!=0
    ↪ and (0,-1)!1<(-1,0)!1 and not isUndefined((1,-1)!1) and
    ↪  (1,-1)!1!=0}
rule : { [(0,0)!0,0,0] }          100 {not isUndefined((-1,1)!1)
    ↪  and not isUndefined((0,1)!1)  and (0,1)!1=0 and (-1,1)
    ↪ !1!=0 and (0,0)!1<(0,0)!0+time/1000   and (0,0)!1!=0
    ↪ and (0,0)!1<(-1,1)!1 and not isUndefined((1,0)!1) and
    ↪ (1,0)!1!=0}

%caso particular: fila de abajo, se puede ir a la derecha
rule : { [(0,0)!0,(0,-1)!1,0] } 100 {not isUndefined((-1,0)!1)
    ↪  and not isUndefined((0,-1)!1) and (0,0)!1=0 and (0,-1)
    ↪ !1!=0 and (0,-1)!1<(0,-1)!0+time/1000 and (-1,0)!1!=0
    ↪ and (0,-1)!1<(-1,0)!1 and isUndefined((1,0)!1)}
rule : { [(0,0)!0,0,0] }          100 {not isUndefined((-1,1)!1)
    ↪  and not isUndefined((0,1)!1)  and (0,1)!1=0 and (-1,1)
    ↪ !1!=0 and (0,0)!1<(0,0)!0+time/1000   and (0,0)!1!=0
    ↪ and (0,0)!1<(-1,1)!1 and isUndefined((1,0)!1) }

%caso particular: gana el de la izquierda porque no hay nada
    ↪ arriba
rule : { [(0,0)!0,(0,-1)!1,0] } 100 {not isUndefined((-1,0)!1)
    ↪  and not isUndefined((0,-1)!1) and (0,0)!1=0 and (0,-1)
    ↪ !1!=0 and (0,-1)!1<(0,-1)!0+time/1000 and (-1,0)!1=0
    ↪ and not isUndefined((1,-1)!1) and (1,-1)!1!=0}
rule : { [(0,0)!0,0,0] }          100 {not isUndefined((-1,1)!1)
    ↪  and not isUndefined((0,1)!1)  and (0,1)!1=0 and (-1,1)
    ↪ !1=0 and (0,0)!1<(0,0)!0+time/1000 and (0,0)!1!=0 and
    ↪ not isUndefined((1,0)!1) and (1,0)!1!=0}

%caso particular: fila de abajo, se puede ir a la derecha
    ↪ porque no hay nada arriba
rule : { [(0,0)!0,(0,-1)!1,0] } 100 {not isUndefined((-1,0)!1)
    ↪  and not isUndefined((0,-1)!1) and (0,0)!1=0 and (0,-1)
    ↪ !1!=0 and (0,-1)!1<(0,-1)!0+time/1000 and (-1,0)!1=0
    ↪ and isUndefined((1,0)!1)}
rule : { [(0,0)!0,0,0] }          100 {not isUndefined((-1,1)!1)
    ↪  and not isUndefined((0,1)!1)  and (0,1)!1=0 and (-1,1)
    ↪ !1=0 and (0,0)!1<(0,0)!0+time/1000   and (0,0)!1!=0
    ↪ and isUndefined((1,0)!1)}
```

```
% caso particular: fila de arriba, se puede ir a la derecha
    ↪ sin preguntar arriba pero hay alguien abajo
rule : { [(0,0)!0,(0,-1)!1,0] } 100 {isUndefined((-1,0)!1) and
    ↪  not isUndefined((0,-1)!1) and (0,0)!1=0 and (0,-1)
    ↪ !1!=0 and (0,-1)!1<(0,-1)!0+time/1000 and not
    ↪ isUndefined((1,-1)!1) and (1,-1)!1!=0}
rule : { [(0,0)!0,0,0] }          100 {isUndefined((-1,1)!1) and
    ↪  not isUndefined((0,1)!1) and (0,1)!1=0 and (0,0)
    ↪ !1<(0,0)!0+time/1000 and (0,0)!1!=0 and not isUndefined
    ↪ ((1,0)!1) and (1,0)!1!=0 }

% trigger de eventos (en la ultima columna no hay)
rule: { [(0,0)!0,(0,0)!1,1] } { ((0,0)!1 - (time/1000 + (0,1)
    ↪ !0))*1000 } {not isUndefined((0,1)!0) and (0,0)!1 >
    ↪ ((0,1)!0 + time/1000) and (0,0)!1!=0}
rule: { [(0,0)!0,(0,0)!1,1] } 0
    ↪                                  {not isUndefined((0,1)
    ↪ !0) and (0,0)!1 < ((0,1)!0 + time/1000) and (0,0)!1!=0}

rule : { (0,0) } 0 { t }
```

Las reglas se agrupan en pares de reglas complementarias asociadas con el movimiento de los productos hacia la derecha y hacia abajo y tienen el siguiente significado:

1. Los primeros 4 pares de reglas están asociados con el movimiento de un producto hacia abajo. Las condiciones del movimiento son las siguientes respectivamente:

   Si una celda está vacía y la celda a su izquierda tiene un producto que de acuerdo con su fecha de vencimiento debe moverse a la columna actual pero que comparando su fecha de vencimiento con la del producto en la celda que está encima de la celda vacía es mayor entonces esta última *gana* la celda vacía.

   Si una celda está vacía y sólo está ocupada la posición de arriba el producto cae.

   Si una celda está vacía y el producto ocupando la posición de la izquierda aún no puede saltar de columna, entonces el producto de arriba cae.

   Si el producto se encuentra en la columna de más a la izquierda, sólo puede bajar.

2. Los siguientes 4 pares de reglas corresponden a un movimiento del producto hacia la derecha, según los siguientes casos:

   Ante una disputa entre la celda de arriba y la de la izquierda, por su fecha de vencimiento gana el producto de la izquierda y la ocupa.

   Mismo caso que antes, pero en la fila de abajo

No hay nada en la celda de arriba a la celda vacía por lo que el producto de la izquierda la puede ocupar libremente, si su fecha de vencimiento le permite pasar de columna.

Mismo caso anterior, pero en la fila de abajo.

3. Esta regla es el caso particular donde la celda vacía se encuentra en la fila de arriba (no es necesario preguntar por arriba) y el producto a la izquierda puede pasar porque tiene alguien abajo que lo "sostiene".

4. Esta regla es la que agenda el momento en que el producto en una celda debe avanzar hacia la derecha. En ese momento se dispara una transición que genera el reacomodamiento del producto. Para esto, simplemente escribe un 1 en la posición 3 de la tupla al momento de vencer el producto (según su columna).

5. La última regla es la que se evalúa siempre positivamente.

Para terminar de definir el modelo, se deben especificar las reglas de comportamiento ante transiciones de los diferentes puertos de entrada. Para esto se plantean dos juegos de reglas. El primero corresponde a las entradas conectadas a la cinta transportadora y debe manejar por un lado la consulta por espacio en la columna correspondiente (valor -1) y por el otro el ingreso de un producto en dicha columna (valor > 0). El segundo corresponde a las consultas que realiza el despachante de productos que consulta por la disponibilidad de un producto en una dada columna del inventario enviando un -1 por el puerto correspondiente. El primer juego de reglas se enuncia a continuación:

```
[prod-top-in]

rule: { [(0,0)!0,(0,0)!1+send(output,(0,0)!1)] } 1 {
    ↪ portValue(thisPort)=-1 }
rule: { [(0,0)!0,portValue(thisPort)] } 1 { portValue(
    ↪ thisPort)!=-1 }
```

Se observa que:

- Si se recibe un valor -1 significa que están preguntando por la disponibilidad de lugar en la columna. Se devuelve el valor de la fecha de vencimiento del producto en la celda superior de la columna por la que se preguntó. Si la celda estaba vacía se devuelve 0.

- Si se recibe un valor > 0 significa que es un producto que viene de la cinta transportadora. Esto significa que previamente se consultó por la disponibilidad de lugar en la columna y se respondió afirmativamente.

```
[query-bot-in]

%borra producto
rule: { [(0,0)!0,0+send(output,(0,0)!1)] } 1 { t }
```

Se observa que:

- Si se recibe una consulta por los puertos de entrada que están debajo del inventario se está consultando por la disponibilidad de un producto para despachar. Entonces se devuelve un producto representado por su fecha de vencimiento: $(0, 0)!1$. Si este valor es > 0 significa que es un producto válido, mientras que si en su lugar se devuelve 0 significa que esa posición del inventario estaba vacía (ya sea porque la columna está vacía o porque esa celda está vacía y aún no se reordenó el inventario).

Este conjunto de reglas con la estructura graficada en la Figura 10 definen el modelo del inventario de productos. Los *links* y la asignación de reglas se pueden ver en la siguiente definición dentro del archivo `inventario.ma`:

```
[inventario]
type : cell
dim : (6,6)
delay : transport
defaultDelayTime  : 0
border : nowrapped

neighbors : inventario(-1,-1) inventario(-1,0) inventario
    ↪ (-1,1)
neighbors : inventario(0,-1)  inventario(0,0)  inventario(0,1)
neighbors : inventario(1,-1)  inventario(1,0)  inventario(1,1)

initialvalue : 0
initialCellsValue : inventario.val
in : pin1 pin2 pin3 pin4 pin5 pin6 qbotin1 qbotin2 qbotin3
    ↪ qbotin4 qbotin5 qbotin6
out : qtopout1 qtopout2 qtopout3 qtopout4 qtopout5 qtopout6
    ↪ pout1 pout2 pout3 pout4 pout5 pout6

link : qbotin6 in@inventario(5,5)
link : qbotin5 in@inventario(5,4)
link : qbotin4 in@inventario(5,3)
link : qbotin3 in@inventario(5,2)
link : qbotin2 in@inventario(5,1)
link : qbotin1 in@inventario(5,0)

link : pin6 pin@inventario(0,5)
link : pin5 pin@inventario(0,4)
link : pin4 pin@inventario(0,3)
link : pin3 pin@inventario(0,2)
link : pin2 pin@inventario(0,1)
link : pin1 pin@inventario(0,0)

link : output@inventario(0,5) qtopout6
link : output@inventario(0,4) qtopout5
link : output@inventario(0,3) qtopout4
link : output@inventario(0,2) qtopout3
link : output@inventario(0,1) qtopout2
link : output@inventario(0,0) qtopout1

link : poutput@inventario(5,5) pout6
link : poutput@inventario(5,4) pout5
link : poutput@inventario(5,3) pout4
link : poutput@inventario(5,2) pout3
link : poutput@inventario(5,1) pout2
link : poutput@inventario(5,0) pout1
```

```
portintransition : pin@inventario(0,5) prod-top-in
portintransition : pin@inventario(0,4) prod-top-in
portintransition : pin@inventario(0,3) prod-top-in
portintransition : pin@inventario(0,2) prod-top-in
portintransition : pin@inventario(0,1) prod-top-in
portintransition : pin@inventario(0,0) prod-top-in

portintransition : in@inventario(5,5) query-bot-in
portintransition : in@inventario(5,4) query-bot-in
portintransition : in@inventario(5,3) query-bot-in
portintransition : in@inventario(5,2) query-bot-in
portintransition : in@inventario(5,1) query-bot-in
portintransition : in@inventario(5,0) query-bot-in

localtransition : inventario-reglas
```

3.4. Despacho de productos

Según lo descripto en la sección 2.2 el despacho de productos es el encargado de extraer los productos del inventario propiamente dicho. Su modelo estructural se muestra en la Figura 13.

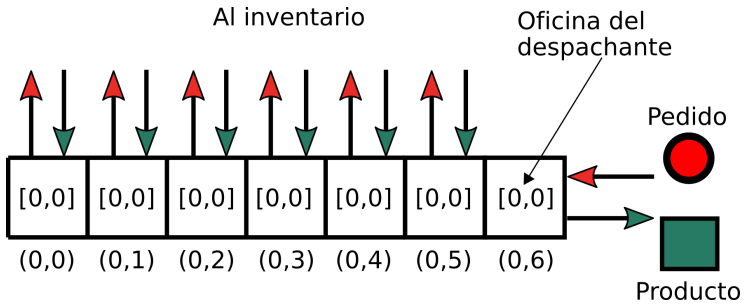


Figura 13: Modelo del despacho de productos.

Conceptualmente se pueden dividir dos tipos de información que se propagarán por el modelo. Por un lado, el despachante que va buscando en qué columna tiene un producto para extraer (recordar que los productos se ordenan de forma que los más próximos a vencer se encuentran cerca de la salida) representado en la Figura 13 por un círculo de color rojo. Por otro, el producto en sí representado por un cuadrado de color verde. Ambos realizarán movimientos en direcciones opuestas, el despachante irá recorriendo el inventario hacia la izquierda buscando la primer columna con productos y, una vez que lo retira, el producto se moverá hacia la derecha en dirección a la salida.

Esta separación da origen a dos grupos (conceptuales) de reglas:

- Reglas para movimiento de productos.

- Reglas para movimiento del despachante.

Ubicadas en ese orden estamos dando prioridad al movimiento de productos y por ende, si un pedido se realiza cuando el despachante no está en su puesto, es decir, cuando está buscando un producto, el pedido se pierde. El vecindario para el despacho de productos se corresponde con el utilizado para la cinta transportadora y se puede observar en la Figura 9.

Se observa en la Figura 13 que el tamaño del autómata celular correspondiente al despacho de productos tiene una celda más que la cantidad de columnas del inventario. Esta celda corresponde a la oficina del despachante en donde se reciben los pedidos de productos. El pedido de un producto se representa con el arribo de un mensaje cuyo contenido es -1.

Se utilizan tuplas de dos elementos para representar el estado de la celda: `[producto, indicador]`. El primer elemento corresponde al producto retirado del inventario, cuando exista un producto en la celda. El segundo elemento permite marcar con -1 cuando se realizó un pedido al inventario y éste informó que en la columna consultada no había disponibilidad, resultando en una consulta fallida. En este último caso el despachante deberá avanzar una columna más hacia la izquierda y volver a consultar al inventario.

Las velocidades del despachante y del producto se ajustaron configurando el tiempo entre evaluaciones de las reglas locales en 100 ms. Mientras que las reglas asociadas con transiciones en las entradas se configuraron con un retardo de 1 ms.

Las reglas asociadas a las transiciones locales se detallan a continuación:

```
[despacho-reglas]
%Movimiento del producto sacado del inventario:
rule : { (0,-1) } 100 { not isUndefined((0,-1)!0) and (0,-1)
    ↪ !0>0 and (0,0)!0=0 }
rule : { [0,0] } 100 { not isUndefined((0,1)!0) and (0,0)!0>0
    ↪ and (0,1)!0=0 }
rule : { [0+send(output,(0,0)!0),0] } 100 { cellPos(1)=6 and
    ↪ (0,0)!0>0 }

%Movimiento del despachante buscando el producto:
rule : {[-1+send(output,-1),0]} 100 { not isUndefined((0,1)!0)
    ↪  and (0,1)=[0,-1] and (0,0)!0=0 }
rule : {[0,0]} 100 { not isUndefined((0,-1)!0) and (0,0)
    ↪ =[0,-1]}

%always true (condicion default)
rule : { (0,0) } 0 { t }
```

Estas reglas se explican a continuación. Las primeras tres reglas son las responsables del movimiento del producto y tienen el siguiente significado:

1. Si la celda contigua a la izquierda existe y contiene un producto (i.e. su valor es > 0) y si la celda actual está vacía, entonces copia el valor de la primera en la segunda.

2. Si la celda contigua a la derecha existe y está vacía entonces el producto se mueve a la derecha (regla anterior) y en el mismo instante la celda actual pasa a 0.

3. Si se alcanzó la última celda de la derecha (oficina del despachante), el elemento `producto` de la tupla se pone en 0 y se envía el producto por la salida de productos.

Las siguientes dos reglas, por su parte, son las responsables del movimiento del despachante y significan respectivamente:

1. Si la celda contigua a la derecha existe y está en estado de consulta fallida: [0,-1], entonces se consulta al inventario por la disponibilidad de producto en la columna asociada a la celda actual (enviando el valor -1) y a esta celda se le asigna el valor [-1,0].

2. A continuación de la regla anterior, se pregunta si la celda actual tiene una consulta fallida y si la celda de la izquierda está definida. En caso afirmativo se asigna a la celda actual el valor [0,0].

Para terminar de definir el modelo, se deben especificar las reglas de comportamiento ante transiciones de los diferentes puertos de entrada. Para esto se plantean dos juegos de reglas. El primero corresponde a la entrada de pedidos y simplemente copia el valor del pedido de productos (-1) en la celda correspondiente (oficina del despachante). El segundo corresponde a las consultas que se envían al inventario, si el inventario devuelve un producto (valor > 0) entonces se copia este valor en el elemento `producto` de la tupla correspondiente a la celda, mientras que si la columna está vacía entonces el inventario devuelve 0 y se asigna a la tupla el valor [0,-1]. Con este último valor se representa a una consulta fallida. Estas reglas se muestran a continuación:

```
[pedidosIn-regla]
rule : { [0,portValue(thisPort)] } 1 { portValue(thisPort)=-1
    ↪ }

[ins-regla]
rule : { [0,-1] } 1 { portValue(thisPort)=0 }
rule : { [portValue(thisPort),0] } 1 { portValue(thisPort)>0 }
```

Este conjunto de reglas junto con la estructura graficada en la Figura 13 definen el modelo del despacho de productos. Los *links* y la asignación de reglas se pueden ver en la siguiente definición dentro del archivo `despacho.ma`:

```
[despacho]
type : cell
dim : (1,7)
delay : transport
defaultDelayTime  : 0
border : nowrapped
neighbors : despacho(0,-1) despacho(0,0) despacho(0,1)
initialvalue : 0
initialCellsValue : despacho.val
in : pedidosIn in1 in2 in3 in4 in5 in6
out : pedidosOut out1 out2 out3 out4 out5 out6
link : pedidosIn in@despacho(0,6)
link : in6 in@despacho(0,5)
link : in5 in@despacho(0,4)
link : in4 in@despacho(0,3)
link : in3 in@despacho(0,2)
link : in2 in@despacho(0,1)
link : in1 in@despacho(0,0)
link : output@despacho(0,6) pedidosOut
link : output@despacho(0,5) out6
link : output@despacho(0,4) out5
link : output@despacho(0,3) out4
link : output@despacho(0,2) out3
link : output@despacho(0,1) out2
link : output@despacho(0,0) out1
portintransition : in@despacho(0,6) pedidosIn-regla
portintransition : in@despacho(0,5) ins-regla
portintransition : in@despacho(0,4) ins-regla
portintransition : in@despacho(0,3) ins-regla
portintransition : in@despacho(0,2) ins-regla
portintransition : in@despacho(0,1) ins-regla
portintransition : in@despacho(0,0) ins-regla
localtransition : despacho-reglas
```

4. Modelado y Simulación

4.1. Pruebas parciales

4.1.1. Cinta Transportadora

Para este ejemplo se utilizó una cinta transportadora de 6 celdas. El tiempo de demora desde que la condición de una regla en las transiciones locales es válida hasta que se ejecuta se configuró en 100 ms, mientras que para las reglas de las transiciones en los puertos de entrada se configuró en 1 ms. Con el primer tiempo se modeliza la velocidad de avance de la cinta transportadora.

Los valores iniciales de las celdas se cargaron mediante el archivo `cinta.val` cuyo contenido se muestra a continuación:

```
(0,0) = [0.6,0   ,0]
(0,1) = [0.5,0   ,0]
(0,2) = [0.4,0   ,0]
(0,3) = [0.3,0   ,0]
(0,4) = [0.2,0   ,0]
(0,5) = [0.1,0.3,0]
```

Como se mencionó anteriormente, se observa que cada celda en la cinta transportadora tiene asociada una tupla de 3 elementos: `[ID columna, producto, indicador]`. El primer elemento de estas tuplas coincide con el rango de fechas de vencimiento de la columna del inventario asociada con la celda de la cinta transportadora. Este rango se mantiene fijo a lo largo de la simulación.

Los eventos de entrada: a) arribo de productos a la cinta y b) respuestas del inventario, se manejan mediante el archivo `cinta.ev` cuyo contenido se muestra a continuación:

```
00:00:00:111 in2 1
00:00:00:221 in3 0
00:05:00:000 in  300.4
00:06:00:000 in3 0
00:10:00:000 in  600.2
00:11:00:000 in2 0
00:15:00:000 in  900.5
00:16:00:000 in3 1
00:17:00:000 in4 1
00:18:00:000 in5 1
00:19:00:000 in6 1
```

Para analizar el comportamiento del autómata celular que utiliza las reglas antes mencionadas y la secuencia de eventos de entrada del archivo `cinta.ev` se ejecutó el simulador `CD++` con el parámetro `-v` de modo de tener una salida *verbosa*. La salida de la prueba es la siguiente:

```
0 / L / Y / 00:00:00:100:0 / out2 / -1.0
0 / L / X / 00:00:00:111:0 /  in2 /  1.0
0 / L / Y / 00:00:00:212:0 / out3 / -1.0
0 / L / X / 00:00:00:221:0 /  in3 /  0.0
0 / L / Y / 00:00:00:221:0 / out3 /  0.3
0 / L / X / 00:05:00:000:0 /   in / 300.4
0 / L / Y / 00:05:00:201:0 / out3 / -1.0
0 / L / X / 00:06:00:000:0 /  in3 /  0.0
0 / L / Y / 00:06:00:000:0 / out3 / 300.4
0 / L / X / 00:10:00:000:0 /   in / 600.2
0 / L / Y / 00:10:00:101:0 / out2 / -1.0
0 / L / X / 00:11:00:000:0 /  in2 /  0.0
0 / L / Y / 00:11:00:000:0 / out2 / 600.2
0 / L / X / 00:15:00:000:0 /   in / 900.5
0 / L / Y / 00:15:00:201:0 / out3 / -1.0
0 / L / X / 00:16:00:000:0 /  in3 /  1.0
0 / L / Y / 00:16:00:101:0 / out4 / -1.0
0 / L / X / 00:17:00:000:0 /  in4 /  1.0
0 / L / Y / 00:17:00:101:0 / out5 / -1.0
0 / L / X / 00:18:00:000:0 /  in5 /  1.0
0 / L / Y / 00:18:00:101:0 / out6 / -1.0
0 / L / X / 00:19:00:000:0 /  in6 /  1.0
```

Esta salida se obtuvo del archivo `log` del modelo *top* reteniendo solamente los eventos de entrada (X) y salida (Y). El primer producto que ingresa en la cinta lo hace durante la inicialización en $t = 00:00:00:000$ (mediante el archivo `cinta.val`) y por ese motivo no aparece aquí. Este primer producto tiene una fecha de vencimiento de 0,3 s. En el tiempo 0 se evalúan todas las celdas y el producto avanza una celda hacia la izquierda alcanzando la celda $(0, 4)$ de la Figura 8. A los 100 ms vuelven a evaluarse las reglas y sólo se ejecutan aquellas celdas que tienen dentro de su vecindario a la celda que sufrió modificaciones en el paso anterior. En este paso el producto no avanza un casillero más hacia la izquierda pues en la comparación de la fecha de vencimiento del producto, `(0,0)!1`= 0,3 s, con el rango de fechas en la columna asociada del inventario, `(0,0)!0 + time/1000`= 0,2 s + 0,1 s, el producto se queda en la celda $(0, 4)$. Entonces en este paso se consulta al inventario si hay lugar disponible en la columna conectada a la salida `out2` asociada a la celda `(0,4)` enviando un -1. Un tiempo después, $t = 00:00:00:111$, llega la respuesta del inventario indicando con un 1 que no hay lugar en esa columna. Entonces el producto avanza una posición hacia la izquierda en la cinta transportadora y se repite la consulta al inventario pero esta vez desde la salida `out3` asociada a la celda `(0,3)`. Al tiempo, $t = 00:00:00:221$, llega la respuesta del inventario indicando con 0 que hay lugar disponible en la columna por la que se consultó e instantáneamente la cinta descarga el producto por la salida. Luego de un tiempo, $t = 00:05:00:000$, ingresa a la cinta transportadora un producto con fecha de vencimiento 300,4 s. Este producto avanza en la cinta hasta la posición $(0, 3)$, cuando `(0,0)!1`= 300,4 s $\leq$`(0,0)!0 + time/1000`= 0,3 s + 300,1 s, consulta si hay lugar en el inventario y se descarga allí en $t = 00:06:00:000$.

De igual manera se procede con el producto que ingresa a la cinta transportadora en $t = 00 : 10 : 00 : 000$ con valor de fecha de vencimiento 600,2 s. Finalmente, el último producto en ingresar a la cinta transportadora de valor 900,5 s lo hace en $t = 00 : 15 : 00 : 000$. Éste avanza hasta la celda $(0, 3)$, cuando `(0,0)!1`$= 900{,}5\text{ s} \leq$`(0,0)!0 + time/1000`$= 0{,}3\text{ s} + 900{,}2\text{ s}$, y consulta al inventario si hay lugar en la columna asociada a esta celda. Ante la respuesta negativa de éste, el producto avanza una posición más en la cinta transportadora y vuelve a consultar. Esta operación se repite y en todos los casos el inventario responde que no tiene lugar para que la cinta descargue el producto, y por lo tanto el producto alcanza el final de la cinta transportadora y permanece allí.

Para tener mayor detalle de la ejecución de las reglas se puede analizar la salida del simulador en modo *verboso*. De allí se puede verificar que en cada paso de simulación sólo se evalúan aquellas celdas que tienen en su vecindario a una celda que sufrió cambios en el paso de simulación anterior. De esta manera no están ejecutándose todo el tiempo todas las reglas para todas las celdas. Esto se puede observar en el siguiente extracto de la salida que corresponde al ensayo anterior (donde para facilitar la legibilidad se quitaron algunas líneas):

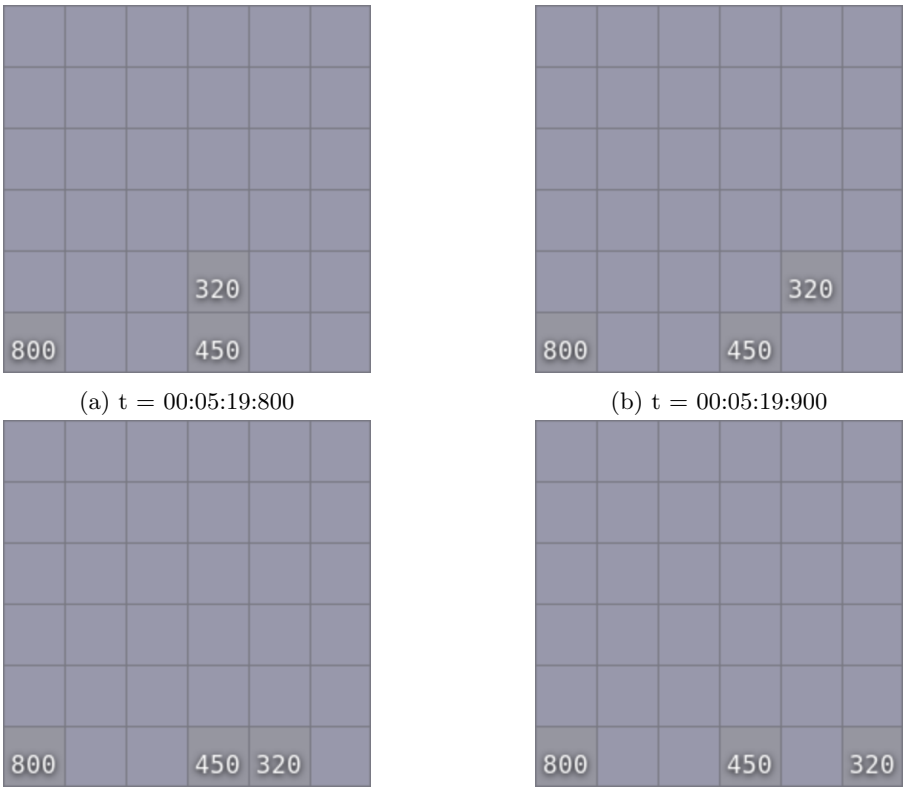
```
+-----------------------------------------------------------+
New Eval: in-regla - 00:05:00:000:0 - cinta(0,5)
Eval: PortIn Reference(in) = 300.4
Eval: SendToNCPort Reference(out, [0.1, 300.4, 0]) at time
    ↪ 00:05:00:000:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:001:0 - cinta(0,4)
Eval: SendToNCPort Reference(out, [0.2, 300.4, 0]) at time
    ↪ 00:05:00:001:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:001:0 - cinta(0,5)
Eval: SendToNCPort Reference(out, [0.1, 0, 0]) at time
    ↪ 00:05:00:001:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:101:0 - cinta(0,3)
Eval: SendToNCPort Reference(out, [0.3, 300.4, 0]) at time
    ↪ 00:05:00:101:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:101:0 - cinta(0,4)
Eval: SendToNCPort Reference(out, [0.2, 0, 0]) at time
    ↪ 00:05:00:101:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:101:0 - cinta(0,5)
Eval: SendToNCPort Reference(out, [0.1, 0, 0]) at time
    ↪ 00:05:00:101:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:201:0 - cinta(0,2)
Eval: SendToNCPort Reference(out, [0.4, 0, 0]) at time
    ↪ 00:05:00:201:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:201:0 - cinta(0,3)
Eval: SendToPort Reference(output, -1) at time 00:05:00:201:0
Eval: SendToNCPort Reference(out, [0.3, 300.4, 0]) at time
    ↪ 00:05:00:201:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:201:0 - cinta(0,4)
Eval: SendToNCPort Reference(out, [0.2, 0, 0]) at time
    ↪ 00:05:00:201:0
+-----------------------------------------------------------+
New Eval: cinta-reglas - 00:05:00:201:0 - cinta(0,5)
Eval: SendToNCPort Reference(out, [0.1, 0, 0]) at time
    ↪ 00:05:00:201:0
+-----------------------------------------------------------+
New Eval: inventario-regla - 00:06:00:000:0 - cinta(0,3)
Eval: PortIn Reference(in3) = 0
Eval: SendToPort Reference(output, 300.4) at time
    ↪ 00:06:00:000:0
Eval: SendToNCPort Reference(out, [0.3, 0, 0]) at time
    ↪ 00:06:00:000:0
```

Se observa el ingreso del producto de fecha de vencimiento 300,4 s por el puerto de entrada en $t = 00 : 05 : 00 : 000$. Este evento modifica el campo `producto` de la tupla en la celda `(0,5)`. Dado que la demora de las reglas asociadas con entradas es de 1 ms, en $t = 00 : 05 : 00 : 001$ se determinan aquellas celdas que se pueden ver afectadas por el cambio en la celda `(0,5)`, que son las celdas `(0,4)` y `(0,5)`, y se evalúa cómo repercute este cambio. A partir de la ejecución de la primera y segunda regla de cada celda el producto es tomado por la celda `(0,4)` y se quita de la `(0,5)`. Las reglas locales se ejecutan cada 100 ms, por lo tanto en el tiempo $t = 00 : 05 : 00 : 101$ se evalúan aquellas celdas que pueden verse afectadas por las celdas que sufrieron cambios en el paso anterior. Estas son las celdas `(0,3)`, `(0,4)` y `(0,5)`. En este paso el producto avanza una celda más hacia la izquierda pasando a la celda `(0,3)`. Nuevamente se esperan 100 ms para volver a evaluar las reglas y en este caso se determina que la celda `(0,3)` es la que le corresponde al producto y entonces se consulta inventario por la disponibilidad de lugar. Se puede observar que como el valor de las celdas evaluadas en $t = 00 : 05 : 00 : 201$ no cambió entonces no se vuelven a evaluar las reglas locales. La respuesta del inventario llega a la celda `(0,3)` en el tiempo $t = 00 : 06 : 00 : 000$ donde indica con 0 que hay lugar disponible en la columna por la que se consultó. A continuación la cinta transportadora descarga el producto de fecha de vencimiento 300,4 s y coloca un 0 en el elemento `producto` de la tupla en la celda `(0,3)`.

4.1.2. Inventario

Las pruebas de este módulo individual se realizaron cargando valores iniciales de las celdas con el archivo `inventario.val`, en conjunto con los eventos configurados en el archivo `inventario.ev` y observando la evolución de los productos en el inventario.

Por medio de la Figura 14 se puede observar el movimiento lateral y hacia abajo de un producto dentro del inventario. Este movimiento depende de si su fecha de vencimiento cae dentro del rango de fechas comprendidas dentro de la columna en la que está ubicado o no, como se explicó anteriormente. Las celdas que no tienen valor están vacías, mientras que las que están llenas muestran las fechas de vencimiento de los producto en ellas. En concreto, se va a analizar el movimiento del producto con fecha de vencimiento 320 s.



(a) t = 00:05:19:800

(b) t = 00:05:19:900

(c) t = 00:05:20:000

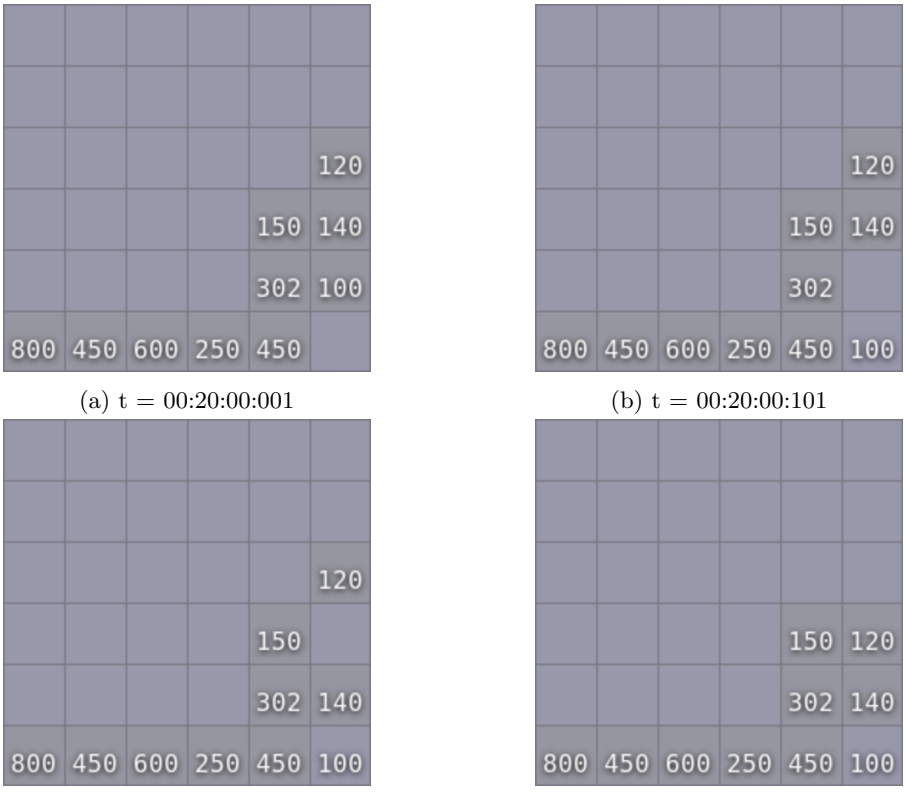
(d) t = 00:05:20:100

Figura 14: Movimiento lateral y hacia abajo de un producto.

Aquí se puede ver el significado de las fechas de vencimiento relativas por columna y absolutas para cada producto. En $t = 00 : 05 : 00 : 000$, que corresponde a 300 s, ingresa a la tercer columna del inventario un producto cuya fecha de vencimiento es 320 s. Éste se desplaza por la columna hacia abajo. Los productos dentro de esta columna son aquellos que vencerán dentro de entre 0,3 s y 0,4 s respecto de la fecha actual, es decir, en el momento en que ingresa el producto al inventario corresponden a fechas de vencimiento entre 300,3 s y 300,4 s. Se puede notar que la fecha de vencimiento del producto es mayor a este rango, sin embargo los productos sólo se puede mover hacia abajo y/o hacia la derecha, por lo tanto si un producto es puesto en una columna equivocada, en la que se deberían ubicar productos con fecha de vencimiento más próxima a la del propio producto, éste permanecerá allí. En $t = 00 : 05 : 19 : 800$, que corresponde a 319,8 s, los productos que pueden estar dentro de la columna son aquellos cuya fecha de vencimiento está comprendida entre: 320,1 s y 320,2 s. Las reglas se evalúan en un momento y se ejecutan luego de 0,1 s de demora, pues con este tiempo modelizamos la dinámica de tiempo asociada con el ma-

nipulador que desplaza los productos dentro del inventario. Por este motivo el producto se desplaza a la columna contigua recién en $t = 00 : 05 : 19 : 900$. En este momento, en la segunda columna se deben alojar aquellos productos con fecha de vencimiento comprendida entre 320,0 s y 320,1 s. Luego de 0,1 s el producto debe desplazarse hacia la siguiente columna a la derecha que para ese momento admitirá aquellos productos cuya fecha de vencimiento sea menor a 320,2 s.

En la Figura 15 se puede observar el comportamiento del inventario ante una situación de disputa. La celda disputada es la que se encuentra en la esquina inferior derecha. Se puede ver que la fecha de vencimiento del producto que está por encima de la celda vacía es menor que la de la celda a su izquierda. Por este motivo, el primer producto gana la disputa y ocupa la celda vacía. A continuación se repite la misma situación para los demás productos que están en la primer columna.



(a) t = 00:20:00:001

(b) t = 00:20:00:101

(c) t = 00:20:00:201

(d) t = 00:20:00:301

Figura 15: Situación de disputa entre 2 celdas.

4.1.3. Despacho de Productos

Los valores iniciales de las celdas de este autómata se cargaron con el archivo `despacho.val`. Para la prueba de este módulo también se procedió a excitarlo con un archivo de eventos `despacho.ev`, que contiene los pedidos de productos y las respuestas del inventario:

```
00:01:00:000 pedidosIn -1
00:01:00:103 in6 0
00:01:00:206 in5 0
00:01:00:309 in4 0
00:01:00:412 in3 300.4
00:01:01:000 pedidosIn -1
00:01:01:103 in6 0
00:01:01:206 in5 0
00:01:01:309 in4 0
00:01:01:412 in3 0
00:01:01:515 in2 0
00:01:01:618 in1 600.2
00:02:00:000 pedidosIn -1
00:02:00:103 in6 900.5
```

Se tuvo especial cuidado en emular las respuestas del inventario en los momentos adecuados de acuerdo con las consultas realizadas al mismo. Consideramos que la asignación de los valores de las reglas tiene un retraso de 100 ms, que modeliza la velocidad del despachante.

Ante este estímulo, el comportamiento que se espera es el siguiente:

1. En el tiempo 00 : 01 : 00 : 000 se recibe un evento de pedido de producto.
2. Luego de 1 ms el pedido se registra en la oficina del despachante (celda `(0,6)`).
3. Luego de 100 ms, el despachante camina hasta la primera columna del inventario (celda `(0,5)`) y hace el pedido al inventario.
4. En el tiempo 00 : 01 : 00 : 103 el inventario responde que no hay productos en esa columna.
5. Luego de 100 ms más el despachante camina una columna más y repite el pedido.
6. Se repite el proceso con sus respectivos tiempos hasta que el despachante llega a la columna 3 (celda `(0,2)`) donde el inventario responde con un producto con fecha de caducidad 600,2 s (recordar que el control de calidad se realiza por un modelo atómico mostrado en el TP1) en tiempo 00 : 01 : 00 : 412.
7. El producto se debe desplazar hacia la derecha a 100 ms por celda hasta salir por el puerto de salida de pedidos.

8. El proceso se repite ante una llegada de un pedido en el tiempo 00 : 01 : 01 : 000. Pero el despachante debe llegar hasta la primer columna para encontrar un producto.

9. Finalmente el proceso se repite en el tiempo 00 : 02 : 00 : 000 pero el inventario dispone de producto en la columna más cercana al despachante (celda (0,5)). Por lo que el recorrido del despachante es solamente de una posición.

Para verificar el comportamiento esperado anterior, se procedió a analizar los archivos `log` generados. Filtrando con el comando `grep`, y conociendo la nomenclatura de los mensajes se puede analizar el comportamiento.

Para la salida del *top-model* podemos observar:

```
Y / 00:01:00:001:0 / out6 /      -1.00000 / top(09)
Y / 00:01:00:104:0 / out5 /      -1.00000 / top(09)
Y / 00:01:00:207:0 / out4 /      -1.00000 / top(09)
Y / 00:01:00:310:0 / out3 /      -1.00000 / top(09)
Y / 00:01:00:813:0 / pedidosout /     300.40000 / top(09)
Y / 00:01:01:001:0 / out6 /      -1.00000 / top(09)
Y / 00:01:01:104:0 / out5 /      -1.00000 / top(09)
Y / 00:01:01:207:0 / out4 /      -1.00000 / top(09)
Y / 00:01:01:310:0 / out3 /      -1.00000 / top(09)
Y / 00:01:01:413:0 / out2 /      -1.00000 / top(09)
Y / 00:01:01:516:0 / out1 /      -1.00000 / top(09)
Y / 00:01:02:219:0 / pedidosout /     600.20000 / top(09)
Y / 00:02:00:001:0 / out6 /      -1.00000 / top(09)
Y / 00:02:00:204:0 / pedidosout /     900.50000 / top(09)
```

Donde se observa que 101 ms después de cada respuesta del inventario se cursa un pedido a la columna siguiente. Por otro lado se observa que el producto sale correctamente por la salida *pedidosOut* 503 ms después de haber entrado en el despacho, tal como fue previsto. Se observa coherencia también en los casos siguientes, con un tiempo de demora en la salida proporcional a la distancia recorrida por el despachante para encontrar un producto en el inventario.

5. Conclusiones

Se comprobó la utilidad del formalismo cell-DEVS para simular el comportamiento de sistemas espaciales que implican dinámicas en varias dimensiones. Se comprobó también la versatilidad del simulador `CD++` que tanto se puede utilizar para simular modelos DEVS como cell-DEVS.

Utilizando reglas simples por celda y replicando estas celdas se pueden lograr comportamientos complejos. Pudimos verificar también que a partir de la definición del vecindario de cada celda se determina qué grupo de celdas evaluar en cada paso de simulación. De esta manera, al igual que sucedía en el TP1 con el formalismo DEVS, verificamos que para sistemas con gran cantidad de celdas esto implica un gran ahorro de cómputo.

A lo largo del desarrollo del trabajo práctico con la herramienta `CD++` en su versión *standalone* se encontraron algunos inconvenientes: a) cuando se incluye un bloque generador se genera un error que indica que el bloque no está registrado, b) cuando se ejecuta el simulador avanzado con el parámetro `-r` se produce una excepción y se aborta la ejecución, c) el archivo de salida generado por el simulador con la opción `-o` genera un archivo vacío, y d) no pudimos definir reglas por zonas ni utilizar *macros*. El primer punto nos llevó a probar todos los autómatas recurriendo a archivos de eventos. Por su parte, el no poder utilizar el comando `cd++ -r` nos dificultó el *debug* de las reglas para cada autómata. Cada vez que invocamos al simulador con esta opción obtuvimos el siguiente mensaje de error:

```
Exception MException thrown!
Description: Rules 8 and 19 evaluate to TRUE and their results
    ↪ are different!
Thrown in:
File: synnode.cpp - Method: evaluate - Line: 118
```

Por esta razón utilizamos el comando `cd++ -v` que devuelve mucha más información y requiere de un análisis más detallado. Por último, el hecho de no poder definir reglas que se ejecuten por zonas dificultó la implementación del autómata celular correspondiente al inventario donde se tuvieron que agregar varias reglas para las filas y columnas en los bordes que tienen un comportamiento distinto a las restantes. Por este mismo motivo, en lugar de haber utilizado 3 automatas celulares para resolver el problema de este trabajo: cinta transportadora, inventario, despacho de productos, podría haberse utilizado tan sólo uno definiendo zonas cada una con sus reglas.

No obstante los comentarios anteriores, la versión avanzada del simulador presenta como ventaja la incorporación de tuplas que simplifican enormemente los vecindarios a utilizar y dan mayor flexibilidad.

Para analizar el resultado del autómata celular correspondiente al inventario utilizamos el visualizador online http://omarhesham.com/arslab/webviewer/ en conjunto con el *script* de python provisto por la cátedra y corregido para seleccionar el elemento de la tupla a visualizar. Sería interesante contar con un visualizador que permita ver más de un elemento de la tupla a la vez.

Al integrar los 3 autómatas celulares implementados en el `topModel.ma` y ensayarlo obtuvimos el siguiente error:

```
Exception AssertException thrown!
Description: Invalid assertion
Thrown in:
        File: pmcoordin.cpp - Method: receive - Line: 262
Description:
        doneCount() > 0
        Unexpected Done message!
```

Ejecutando el simulador en modo *verboso* podemos ver que la excepción se debe a un error de **rollback**:

```
StateVars: ()
```

```
Rolling back, current time: 00:00:00:100:0 last time:
    ↪ 00:01:00:000:0
Aborting simulation...
```

Por esta razón sólo pudimos ensayar cada autómata celular de forma individual.

A. Código Implementado

https://github.com/TwinT/DEVS-Inventario