[75.26] Simulación

María Inés Parnisari

17 de diciembre de 2014

1. Definiciones

Sistema: conjunto de elementos que interactúan entre si, que se aíslan del universo para su estudio.
Modelo: representación abstracta de un sistema real.

- Se puede demostrar su **invalidez** con una no concordancia entre los resultados obtenidos en el modelo y los obtenidos en el sistema real.
- Deben estar identificadas las **entidades** y sus atributos:
  - Entidades **permanentes**: *savevalues, functions, variables, SNAs, facilities, storages, queues, tables*
  - Entidades **transitorias**: transacciones
- Debe incluir **variables**:

Endógenas	**Exógenas**
Internas, controladas por el sistema	Externas, no controladas por el sistema
Cualitativas	**Cuantitativas**
Preferencia personal	Frecuencia con que arriban clientes

Cuadro 1: Tipos de variables

- Compuesto por **bloques** que representan las acciones que realizan las **transacciones** en el sistema.

Numérico	**Analítico**
Se conoce instante a instante el valor de las variables, pero no la relación que hay entre ellas	Se conoce la relación entre las variables
Matemático	**Físico**
Representación abstracta	Representación tangible
Dinámico	**Estático**
Su estado varía en el tiempo	Su estado no varía en el tiempo
Discreto	**Continuo**
Los cambios se producen de a saltos	Los cambios se producen gradualmente
Estocástico	**Determinístico**
Existe azar en la ocurrencia de eventos	No existe azar en la ocurrencia de eventos

Cuadro 2: Tipos de modelos

Simular: ensayar en un modelo una alternativa para inferir lo que pasaría en el sistema real si se aplica dicha alternativa. Es predecir el futuro ante hipótesis ciertas.

- Razones por las que puede fallar una simulación:
  1. Modelo inválido: no representa fielmente al sistema real
  2. Las alternativas simuladas no son buenas
- Objetivo: predicción
- Fases:
  1. Definir el sistema (aislar las partes a estudiar)
  2. Construir el modelo (representar el sistema)
  3. Simular (ensayar alternativas)
  4. Sacar conclusiones
  5. Estudiar las conclusiones y aconsejar la mejor alternativa
- Ventajas:
  - Permite adquirir una rápida experiencia a bajo costo y sin riesgos,
  - Permite identificar áreas con problemas en sistemas complejos,
  - Permite un estudio sistemático de alternativas,
  - Es ilimitado en cuanto a complejidad.
- Desventajas:
  - No supera a las técnicas analíticas,
  - No se puede asegurar que el modelo es válido,

- No existe un criterio científico de selección de la estrategia de simulación,
- Existe un riesgo de utilizar un modelo fuera de los límites para el cual fue construido.

Estrategia de simulación: conjunto de alternativas que se definen para un ensayo.

Evento: acción que provoca un cambio de estado, modificando los atributos de las entidades.

2. Cadenas

Las transacciones son temporariamente ligadas a entidades ocupando listas vinculadas llamadas cadenas.

- La **Cadena de Eventos Corrientes** (CEC) es una lista encadenada de transacciones preparadas que ya tienen bloques para ser ingresados antes que la simulación avance. Cuando la transacción activa descansa en alguna cadena, la primer transacción de mayor prioridad en la CEC se vuelve la transacción activa. Si la CEC esta vacía, la transacción(s) más inminente en la Cadena de Eventos Futuros (FEC) es o son movidas a la CEC. Esta acción siempre causa un avance en el sistema de reloj.

  La ubicación de la CEC puede ser modificada por los bloques PRIORITY y BUFFER.

- La **Cadena de Eventos Futuros** (FEC) es una cadena ordenada en el tiempo que retiene transacciones que deben esperar por un tiempo de simulación. Cuando toda actividad de la simulación durante el tiempo de reloj corriente es completada, la siguiente transacción es tomada de la FEC. Los bloques ADVANCE y GENERATE son la única manera de colocar una transacción en la FEC.

- Cada entidad tiene una cadena de transacciones bloqueadas, llamada **Cadena de Reintento** (RETRY CHAIN). Las transacciones que fallan todos los testeos requeridos para entrar a un bloque son colocadas en una Cadena de Reintento. Estos testeos ocurren cuando una transacción intenta entrar en un bloque GATE o TEST.

- Las **Cadenas de Usuario** (USER CHAIN) son listas encadenadas de transacciones que han sido removidas de la Cadena de Eventos Corrientes por un bloque LINK.

3. Funcionamiento del GPSS

```
por cada GENERATE del modelo:
  crea una transacción
  pone la transacción en la FEC (Future Event Chain)

mientras (cantidad de transacciones destruidas < parámetro n del START)
    ordena la FEC en orden cronológico

    primer evento := FEC [0]
    reloj := hora del primer evento

    para cada transacción de la FEC:
      si (hora de transacción == reloj)
        agregar transacción a CEC (Current Event Chain)

    ordenar la CEC por prioridad

    para cada transacción de la CEC:
      mover la transacción por el modelo hasta que:
      {
        si (transacción ejecuta un LINK):
          transacción queda esperando
```

```
      si (transacción quiere tomar un recurso no disponible):
        poner transacción en DELAY CHAIN del recurso
      si (transacción atraviesa una condición falsa sin alternativas):
        poner transacción en RETRY CHAIN de la condición
      si (transacción quiere arrebatar un recurso y no puede):
        poner transacción en PENDING CHAIN del recurso
      si (transacción quiere hacer ADVANCE con tiempo != 0):
        poner transacción en FEC con el instante de finalización
      si (transacción muere):
        quitar transacción de CEC
      si (transacción es la primera de un ASSEMBLE):
        transacción queda esperando
      si (transacción genera nueva transacción X)
        poner X en CEC
    }
```

4. Bloques y objetos

Bloque: representa las acciones que realizan las transacciones en el sistema.

- Objetos transitorios
  - **Transacción**: elemento que se mueve por el modelo.
    - Atributos: se instancian la primera vez que se les asigna valor. (`ASSIGN`, `SELECT`, `MARK`, `ALTER`).
      - ◇ **Prioridad**: 0 (mínima), 100 (máxima). Se puede alterar con los bloques `PRIORITY` y `ALTER`, y examinar con el bloque `SCAN`. Se inicializa en `GENERATE`.
      - ◇ `M1`: edad, se inicializa en cero al nacer la transacción, y se puede alterar con el bloque `MARK`. Al momento de consultar el `M1`, se hace la resta entre la hora actual y la hora de nacimiento.
      - ◇ **Parámetros**: son locales a la transacción. Se modifican con `ASSIGN` y `SELECT`.
    - Cada transacción tiene un **puntero** que apunta al primer primer miembro de la familia, que, al nacer cada transacción, apunta a sí misma (cada transacción creada por `GENERATE` pertenece a familias independientes entre sí).
- Objetos permanentes:
  - Recursos definidos explícitamente: `STORAGE`, `TABLE`, `QTABLE`.
  - Recursos definidos implícitamente: `FACILITY`, `QUEUE`.

Bloques que generan transacciones: `GENERATE`, `SPLIT`.
Bloques que colocan transacciones en la FEC (*Future Event Chain*): `GENERATE`, `ADVANCE`.
Bloques que destruyen transacciones: `TERMINATE`, `ASSEMBLE`.
Bloques que bloquean transacciones si no existe rótulo alternativo y la proposición es falsa: `TEST`, `GATE`.
SNAs:

- `AC1`: hora absoluta, en segundos. Se inicializa en cero al comenzar la simulación. Es el tiempo simulado desde el último *clear*.
- RN3, RN4, RN5...: número al azar entre 0 y 999.

- C1: hora relativa, en segundos. Es el tiempo simulado desde el último *reset*.

Facility	**Storage**
`SEIZE, RELEASE`	`ENTER, LEAVE`
Capacidad: 1	Capacidad: n
Es obligatorio que la transacción que ejecuta el `RELEASE` sea la que anteriormente ejecutó `SEIZE`.	No es obligatorio que la transacción que ejecuta el `LEAVE` sea la que anteriormente ejecutó el `ENTER`.
Es arrebatable.	No es arrebatable.
Cadena de Demora (DELAY CHAIN) – Una cadena de prioridad de transacciones esperando por la propiedad de una *facility*. Cadena de Pendientes (PENDING CHAIN) – Una lista de transacciones esperando por apropiarse de la *facility*. Cadena de Interrupción (INTERRUPT CHAIN) – Una lista de transacciones que han sido apropiadas desde la propiedad de la *facility*. Cadena de Reintento (RETRY CHAIN) – Una lista de transacciones que están esperando por el cambio de estado de una *facility*.	Cadena de Demora (DELAY CHAIN) – Una cadena de prioridad de transacciones esperando por una unidad *storage*. Cadena de Reintento (RETRY CHAIN) – Una lista de transacciones que están esperando por el cambio de estado del *storage*.

Cuadro 3: Diferencia entre *storages* y *facilities*

	Parámetro	***Savevalue***	***Logic switch***
¿Localidad?	Pertenece a una transacción	Global	Global
¿Valor posible?	Cualquier valor	Cualquier valor	2 valores - set (1), reset (0)
¿Valor inicial?	-	0	reset (0)

Cuadro 4: Diferencia entre parámetros, *savevalues* y *logic switches*

5. Cadenas de usuario

Cadena: objeto donde las transacciones se "encadenan" de modo tal que si otra transacción no las libera, jamás saldrán de allí.
Permiten representar interacciones entre dos subsistemas.

- Transporte de transacciones: la transacción **activa** realiza el transporte, y la transacción **pasiva** es transportada.
- Recursos que no siempre están disponibles: se los representa como una transacción y como una *facility*.

- Impaciencia de clientes: se subdivide al cliente en dos partes; uno es el original, y la copia es la transacción que espera un tiempo fijo.
  - Impaciencia de grado 1: por ejemplo, si la cantidad de gente en cola supera 8, me voy de la cola.
  - Impaciencia de grado 2: por ejemplo, si el tiempo que pasé en cola supera 1 hora, me voy de la cola.

```
1      GENERATE     clientes
2      TEST  LE  Q$JUAN,7,CHAU     ;si en la cola de Juan hay mas de 7 clientes, va a CHAU
3      SAVEVALUE  CUEN+,1
4      ASSIGN     4,X$CUEN         ;P4 tiene el numero de cliente
5      SPLIT    1,IMPA         ;generamos una copia y la mandamos a IMPA
6      QUEUE    JUAN           ;hace cola para el facility Juan
7      LINK       JUAN,FIFO
8  ATIEN   SEIZE     JUAN
9      DEPART     JUAN
10     ADVANCE    tiempo de atencion
11     RELEASE    JUAN
12     TERMINATE
13 IMPA  ADVANCE     tiempo de paciencia  ;soy la copia que espera un tiempo
14     UNLINK     JUAN,SEVA,1,4,*4     ;me impacienté: desencadeno a 1 transacción cuyo P4 sea igual al P4 mío
15     TERMINATE              ;destruyo la copia
16 SEVA  DEPART       JUAN
17 CHAU  TERMINATE
```

6. Cartilla de bloques

Id: número o nombre.

Bloque		Acción que representa	Parámetros	Default
`START`	`n`	Simular n transacciones	n = cantidad de transacciones	
`GENERATE`	`[t],[d],[o],[c],p]`	Arribo de transacciones independientes entre sí cada cierto tiempo $t \pm d$ o $t \times d$	▪ t = valor medio entre arribos ▪ Si $d \in N$, es el desvío máximo permitido. Si d es una `FUNCTION`, se multiplica $t \times d$ para calcular el próximo nacimiento ▪ o = instante en que se genera la primera transacción ▪ c = cantidad máxima de transacciones a generar ▪ p = prioridad de la transacción a generar	▪ $t = 0$ ▪ $d = 0$ ▪ $o = 1$ ▪ $c = \infty$ ▪ $p = 0$ (prioridad mínima)
`ADVANCE`	`t[,d]`	Realización de una tarea que dura cierto tiempo $t \pm d$ o $t \times d$	▪ t = valor medio de la duración de la tarea (constante o SNA) ▪ Si $d \in N$, es el desvío máximo permitido. Si d es una `FUNCTION`, se multiplica $t \times d$ para calcular la duración. *No puede ser referencia a FUNCTION.* ▪ Debe cumplirse que $t \geq d$	▪ $t = 0$ ▪ $d = 0$
`TERMINATE`	`[n]`	Salida de n transacciones. Si n no se especifica, no se contabiliza la transacción pero la misma se destruye	$n \in N$	$n = 0$
`SEIZE`	`f`	Tomar un recurso de uso exclusivo (*facility*)	f = id de la *facility*	
`RELEASE`	`f`	Dejar un recurso de uso exclusivo (*facility*)	f = id de la *facility*	

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
	ENTER	s[,l]	Intentar ocupar uno o más lugares de un recurso de uso compartido (*storage*)	▪ s = id del *storage* ▪ l = lugares que ocupa la transacción	$l = 1$
	LEAVE	s[,l]	Dejar uno o más lugares de un recurso de uso compartido (*storage*). No es necesario haber ejecutado ENTER previamente.	▪ s = id del *storage* ▪ l = lugares que deja la transacción	$l = 1$
NSTORAGE	STORAGE	n	Definir la capacidad de un *storage* en n transacciones	▪ $NSTORAGE$ = identificador del *storage* ▪ $n \in N$, $n < 30000$	
	TRANSFER	[p,[a]],b	Bifurcar estocásticamente. Con probabilidad p se bifurca a b, y con probabilidad $1 - p$ se bifurca a a. Si a no se especifica, bifurca al bloque siguiente con probabilidad $1 - p$. Si ni p ni a se especifican, bifurca siempre a b	▪ $0 \leq p < 1$, o p es una referencia a una función de n puntos tal que $0 \leq y_i < 1000$ $(i = 1..n)$ ▪ a y b son rótulos o referencias a funciones	$p = 1$
	TRANSFER	SIM,b,c	Si el *delay indicator* de la transacción está en ON (set) —si alguno de los GATE/TEST anteriores no se cumple—, la transacción es enviada al rótulo c y pone el *delay indicator* en OFF (reset). Si el *delay indicator* está en OFF —todos los GATE/TEST anteriores se cumplen **simultáneamente**—, la transacción es enviada al rótulo b.	b y c son rótulos.	

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
	QUEUE	f[,l]	Hacer cola para la *facility* f	▪ f = id de la *queue* ▪ l = lugares que ocupa la transacción	$l = 1$
	DEPART	f[,l]	Salir de la cola de la *facility* f, si la QTABLE respectiva está definida, registra allí las estadísticas	▪ f = id de la *queue* ▪ l = lugares que deja la transacción	$l = 1$
	START	n	Simular en el modelo hasta contabilizar n transacciones terminadas	$n \in N$	
NTABLE	TABLE	v,x_0,Δx,n	Definir una tabla de distribución de frecuencia. Necesita de TABULATE.	▪ v = valor a tabular de la transacción que ejecuta el *tabulate* (cualquier SNA) ▪ x_0 = Límite superior del primer intervalo ▪ Δx = tamaño de cada intervalo ▪ n =cantidad de intervalos	
	TABULATE	t	Agrega un valor a una TABLE	t = id de la TABLE	
NQTABLE	QTABLE	ntable,t_0,Δt,n	Define una tabla de distribución de frecuencias para tiempos en cola. No necesita de TABULATE.	▪ $NQTABLE$ = id de la *queue* ▪ t_0 = límite superior del primer intervalo ▪ Δt = tamaño de cada intervalo ▪ n = cantidad de intervalos	
	ASSIGN	p,v p+,v p-,v	Asigna, suma o resta el valor v al parámetro p de la transacción activa	p = id del parámetro (puede ser un SNA) v = valor a asignar, sumar o restar (puede ser una constante o un SNA)	

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
SELECT	op logico[1]	p,v_1,v_2[,,,f]	Busca una *facility, storage* o *logic switch* según el op logico y la almacena en el parámetro p de la transacción activa. Al finalizar, $v_1 \leq a \leq v_2$ o, si no pudo seleccionar, bifurca a f. Si f no esta, establece que $p = 0$ y continúa en el bloque siguiente	▪ p = id del parámetro ▪ v_1 = número de objeto menor desde el cual se hace el SELECT ▪ v_2 = número de objeto mayor desde el cual se hace el SELECT ▪ f = rótulo al que bifurca si la selección fue infructuosa	
SELECT	op relacion[2]	a,v_1,v_2,d,e[,f]	Busca una objeto e según el op relacion y lo almacena en el parámetro a de la transacción activa. Al finalizar, $v_1 \leq a \leq v_2$ o, si no hay desocupadas, bifurca a f. Si f no esta, establece que $a = 0$ y continúa en el bloque siguiente	▪ a = id del parámetro ▪ v_1 = número de objeto menor desde el cual se hace el SELECT ▪ v_2 = número de objeto mayor desde el cual se hace el SELECT ▪ d = valor con el que se compara ▪ e = clase de objeto que se compara (SNAs) ▪ f = rótulo al que bifurca si la selección fue infructuosa	

[1] Logic Switch: LS, LR
Facility: U, NU, I, NI, FV, FNV
Storage: SE, SNE, SF, SNF, SV, SNV
[2] EQ, L, LE, G, GE, NE

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
SELECT	MIN/MAX	p,v_1,v_2,,e	Busca un *facility* desocupado según el criterio e y lo almacena en p. Si el criterio no alcanza para definir, $p = v_1$	▪ p = id del parámetro ▪ v_1 = número menor del objeto a consultar ▪ v_2 = número mayor del objeto a consultar ▪ e = clase de objeto en la que se busca el mínimo o máximo	
NVARIABLE	VARIABLE	f	Define una función que se evalúa cada vez que una transacción hace referencia a ella. Se truncan los decimales de los resultados intermedios y del resultado final.	f = función con operadores (cualquier SNA) + suma - resta # multiplicación / división @ módulo ^ potencia	
NFUNCTION	FUNCTION	x,tn x_1,y_1/x_2,y_2/.../x_n,y_n	Devuelve un valor cada vez que se la invoca. Define una función de n puntos. Si $x \leq x_1$, devuelve y_1, si $x \geq x_n$, devuelve y_n. Se truncan los decimales de los resultados intermedios y del resultado final.	▪ x = variable independiente (cualquier SNA). Si x es RN, $0 < x_i \leq 1$ ▪ t = tipo de la función (C =continua, D =discreta, E =discreta de atributos numéricos)[3] ▪ n = cantidad de puntos que definen la función	
	SAVEVALUE	s,v s+,v s-,v	Asigna, suma o resta el valor v al *savevalue* s	▪ s = id del *savevalue* ▪ v = valor a asignar, sumar o restar	
	INITIAL	s[,a]	Asigna el valor inicial a al *savevalue* s	s = **referencia** al *savevalue*	?

[3]Debe cumplirse que $x_1 \leq x_2 \leq ... \leq x_n$.
Si es **continua**, y $x_i < x < x_{i+1}$ se interpola linealmente entre y_i e y_{i+1}. Si es **discreta**, y $x_i < x < x_{i+1}$, se devuelve y_{i+1} (los y pueden ser rótulos). Si es **discreta de atributos numéricos**, es como la discreta pero las y deben ser SNAs.

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
	LOOP	p,a	Resta 1 al parámetro p. Si $p > 0$, bifurca al rótulo a. Si $p = 0$, continúa en el bloque siguiente	▪ p = id del parámetro a decrementar ▪ a = rótulo al que bifurca	
NMATRIX	MATRIX	,f,c	Define una matriz	▪ f = cantidad de filas ▪ c = cantidad de columnas	
	MSAVEVALUE	m,f,c,v m+,f,c,v m-,f,c,v	Asigna, suma o resta el valor v del elemento $m[f, c]$	▪ m = id de la matriz ▪ f = fila (cualquier SNA) ▪ c = columna (cualquier SNA) ▪ v = valor a asignar, sumar o restar	
TEST	op	a,b[,f]	Si $a\,op\,b$ es V, continúa en el bloque siguiente. Si es F, bifurca al rótulo f; si es F y no se especifico f, bloquea la transacción hasta que sea V	▪ op puede ser e, ne, g, ge, l, le ▪ a y b pueden ser constantes o SNAs ▪ f = rótulo al que bifurca	
	SPLIT	c,r[,n]	Genera c nuevas transacciones, "clonando" la transacción activa (M1, PR, parámetros). Las copias bifurcan a r y el original continúa en el bloque siguiente	▪ c = cantidad de copias (constante o SNA) ▪ r = rótulo al que bifurcan las copias ▪ n = id de parámetro a utilizar para enumerar el original y las copias	
	ASSEMBLE	a	Reúne un grupo de transacciones y deja pasar solo una transacción que representa al grupo, las restantes $a - 1$ son destruidas	a = cantidad de miembros de la familia a sincronizar	

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
	GATHER	m	Reúne un grupo de transacciones y deja pasar a todas una vez reunida la cantidad especificada, sin destruir ninguna transacción	m = cantidad de miembros de la familia a reunir	
	MARK	[p]	Se almacena un M1 "paralelo" en el parámetro p de la transacción. Si p no se especifica, se inicializa M1 en cero.	p = id del parámetro	$p = M1$
	PRIORITY	p	Cambia la prioridad de la transacción activa	p = nueva prioridad	
LOGIC	v	ll	Cambia el estado del *logic switch ll* por el valor de v	▪ v = S (set), R (reset), I (invertido) ▪ ll = id de la llave	
	INITIAL	rll[,v]	Inicializa el estado de la llave rll con el valor v.	▪ rll = **referencia** a una llave lógica ▪ v = 1 (set), 0 (reset)	$v = 1$(set)
GATE	op lógico4	a[,f]	Consulta el estado del *logic switch*, *facility*, *storage* o rótulo a. Si opA es V, continúa en el bloque siguiente. Si es F, bifurca a f. Si no se especifica f, la transacción se bloquea hasta que sea V (*delay indicator* en ON)	▪ a = id del *logic switch*, *facility*, *storage* ▪ f = rótulo al que bifurca si la condición es falsa.	
COUNT	op logico[4]	a,v_1,v_2	Cuenta la cantidad de objetos desde v_1 hasta v_2 que cumplen la condición $opobjeto$, y lo almacena en a.	▪ a = parámetro que recibe la cantidad de objetos que cumplen la condición ▪ v_1 = número menor del objeto a consultar ▪ v_2 = número mayor del objeto a consultar	

[4]Logic Switch: LS, LR
Facility: U, NU, I, NI, FV, FNV

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
COUNT	op relacion[5]	a,v_1,v_2,d,e	Cuenta la cantidad de objetos desde v_1 hasta v_2 que cumplen la condición *objeto op d*, y lo almacena en a.	▪ a = parámetro que recibe la cantidad de objetos que cumple la condición ▪ v_1 = número menor del objeto a consultar ▪ v_2 = número mayor del objeto a consultar ▪ d = valor con el que se compara ▪ e = clase de objeto que se selecciona (SNA)	
COUNT	MIN/MAX	a,v_1,v_2,,e	Cuenta la cantidad de objetos desde v_1 hasta v_2 que son mínimo o máximo, y lo almacena en a.	▪ a = parámetro que recibe la cantidad de objetos que son mínimo o máximo ▪ v_1 = número menor del objeto a consultar ▪ v_2 = número mayor del objeto a consultar ▪ e = clase de objeto que se compara (SNA)	
	JOIN	g	La transacción activa se une al grupo g "personalmente".	g = id del grupo.	
	EXAMINE	g,,f	Verifica si la transacción activa pertenece al grupo g. Si pertenece, continúa en el bloque siguiente. Si no, bifurca a f.	▪ g = id del grupo. ▪ f = rótulo al que bifurca.	
	REMOVE	g	La transacción activa se elimina al grupo g "personalmente".	g = id del grupo.	

Storage: SE, SNE, SF, SNF, SV, SNV

[5] EQ, L, LE, G, GE, NE

Bloque			**Acción que representa**	**Parámetros**	**Default**
	START	n	Simular n transacciones	n = cantidad de transacciones	
REMOVE	[op][6]	g,[c],,[p],[v],[f]	Intenta eliminar c transacciones del grupo g que cumplen la condición $p\,op\,v$. Si no puede eliminar todas, bifurca a f.	▪ g = id del grupo ▪ c =cantidad de transacciones a eliminar ▪ p = idel parámetro que se consulta de la transacción ▪ v =valor con el que se compara (no se pone si $op = MIN/MAX$) ▪ f =rótulo al que bifurca si no pudo eliminar c transacciones	$op = E$ $c = ALL$
ALTER	[op][7]	g,c,p_1,v_1,[p_2],[v_2],[f]	Intenta alterar c transacciones del grupo g de la siguiente forma: para cada transacción, si $p_2\,op\,v_2$ es V, se asigna $p_1 = v_1$. Si no puede alterar c transacciones, bifurca al rótulo f.	▪ g = id del grupo ▪ c = cantidad de transacciones a eliminar[8] ▪ p_1 = id del parámetro a alterar ▪ v_1 =valor a asignar al parámetro p_1 (constante o SNA) ▪ p_2 = id del parámetro a consultar ▪ v_2 = valor con el que se compara p_2 (constante o SNA) ▪ f =rótulo al que bifurca si no pudo alterar c transacciones	$op = E$

[6] E,G,GE,L,LE,MAX,MIN,NE
[7] E,G,GE,L,LE,NE
[8] c puede ser ALL

Bloque			**Acción que representa**	**Parámetros**	**Default**
	`START`	`n`	Simular n transacciones	n = cantidad de transacciones	
`SCAN`	`[op]`	`g,b,[c],`p_1`,`p_2`[,f]`	Busca en el grupo g la **primera** transacción que cumple la condición $b\,op\,c$. Copia el parámetro p_1 en el parámetro p_2 de la transacción actual. Si no encuentra ninguna, bifurca a f.	▪ OP puede ser $e, g, ge, l, le, max, min, ne$ ▪ b = id del parámetro que se consulta ▪ c = valor con el que se compara (si OP es min/max, no se pone) ▪ p_1 = id del parámetro de la transacción del grupo que se copia ▪ p_2 = id del parámetro de la transacción actual que recibe el valor de p_1	▪ $op = E$
	`LINK`	`c,r[,f]`	Encadena la transacción pasiva actual a la cadena c según la regla r. Si se especifica el rótulo f, la primera transacción **no es encadenada** y bifurca al rótulo f.	▪ c = id de la cadena (constante o SNA) ▪ r = regla por la cual se encadena (FIFO, LIFO, PR, Pn, M1) ▪ f = rótulo al que va la primera transacción	

Bloque			Acción que representa	Parámetros	Default
	START	n	Simular n transacciones	n = cantidad de transacciones	
UNLINK	[op]	a,b,c,[d],[e],[f]	La transacción activa actual desencadena c transacciones de la cadena a tales que se verifique $d\,op\,e$, y las envía al rótulo b. Si no puede liberar todas, bifurca a f (las que fueron liberadas, quedan liberadas).	▪ op puede ser E,G,GE,L,LE,NE ▪ a = id de la cadena ▪ b = rótulo al que se envían las transacciones liberadas ▪ c = cantidad de transacciones que se desencadenan[9] ▪ d = id de parámetro de la transacción **activa** que se consulta ▪ e = **referencia** del parámetro de la transacción **pasiva** que se consulta ▪ f = rótulo al que bifurca la transacción activa si no pudo liberar c transacciones.	▪ $op = E$
	BUFFER		La transacción actual se coloca como última transacción a mover en la CEC. Detiene a la misma sin que pase el tiempo.		
	PREEMPT	a	La transacción actual arrebata la *facility* a que ha sido tomada por un SEIZE, sin importar las prioridades. Si fue tomada por otro PREEMPT, **no** la arrebata. Interrumpe el ADVANCE del *owner* de la *facility*.	a = id de *facility* a intentar arrebatar	
	PREEMPT	a,PR[,c,d[,RE]]	El arrebato se produce sólo si el *owner* de la *facility* a tiene menos prioridad que la transacción actual. Si se especifica RE, el arrebato es definitivo.	a = id de *facility* a intentar arrebatar c = rótulo al que bifurca el *owner* si estaba ejecutando un ADVANCE d = id del parámetro del *owner* donde se almacena el tiempo remanente del *owner* para terminar el ADVANCE	
	RETURN	a	Retorno de recurso a al *owner* de dicha *facility*		

[9] c puede ser ALL

Bloque		Acción que representa	Parámetros	Default
START	n	Simular n transacciones	n = cantidad de transacciones	
FUNAVAIL	a,[b],[c],[d],[e],[f],[g],h	Rotura de *facility* a. El owner de la *facility* bifurca al rótulo c. Las transacciones que fueron interrumpidas bifurcan al rótulo f. Las transacciones que están en la *delay chain* o *pending chain* de a bifurcan al rótulo h.	▪ a = id de *facility* ▪ $b = RE$ (el *owner* de la *facility* la deja definitivamente) o CO (el owner continúa en poder de la *facility*) ▪ c es obligatorio si $b = RE$ ▪ d = parámetro del *owner* de la *facility* donde se guarda el tiempo remanente del ADVANCE que estuviera ejecutando. ▪ $e = RE$ (las transacciones interrumpidas la dejan definitivamente) o CO (el owner continúa en poder de la *facility*) ▪ f es obligatorio si $e = RE$ ▪ $g = RE$ (las transacciones en la *delay chain* o *pending chain* abandonan la idea de tomar a) o CO (abandonan la *delay chain* o *pending chain*) ▪ si h es usado, g debe ser RE	
FAVAIL	a	Arreglo de *facility* a		
SUNAVAIL	a	Rotura de *storage* a		
SAVAIL	a	Arreglo de *storage* a		
MATCH	a	Sincronizar el movimiento de transacciones de la misma familia.	a = rótulo para ser testeado por transacciones iguales.	
JOIN	g,n	Unir el número n al grupo numérico g	▪ g= id del grupo **numérico** o SNA ▪ n =constante o SNA	
REMOVE	g,,n[,,,f]	Remover al número n del grupo numérico g. Si no pertenece, bifurca a f	▪ g= id del grupo **numérico** o SNA ▪ n =constante o SNA	

Bloque		Acción que representa	Parámetros	Default
`START`	`n`	Simular n transacciones	n = cantidad de transacciones	
`EXAMINE`	`g,n,f`	Si el número n pertenece al grupo numérico g, continúa en el bloque siguiente. Si no, bifurca a f	▪ g= id del grupo **numérico** o SNA ▪ n =constante o SNA	

Para referirse a parámetros:

1. Por nombre:
   a) `P$CAJAS`
   b) `*$CAJAS`
   c) `*CAJAS`
2. Por número:
   a) `P4`
   b) `*4`

Para referirse a un `SAVEVALUE`:

1. Por nombre: `X$CAJAS`
2. Por número: `X4`

Para referirse a una `VARIABLE`:

1. Direccionamiento directo: `V$CAJA, V$1`
2. Direccionamiento indirecto: `V*CAJA, V*1`

Para referirse a una `FUNCTION`: `FN$FUNCION`
Para referirse a una `MATRIX`:

1. Por nombre: `MX$MATRIZ(col,fil)`
2. Por número: `Mx4(col,fil)`