Aplicaciones de modelos celulares complejos usando N-CD++

Alumno:
Javier Ameghino LU 316/90

Director:
Dr. Gabriel Wainer

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

2000

Tabla de Contenidos

1. RESUMEN 5

2. INTRODUCCIÓN 5

2.1 FORMALISMO DEVS 6
2.1.1 Modelos Atómicos 6
2.2 AUTÓMATAS CELULARES 8
2.3 FORMALISMO CELL–DEVS 11

3. HERRAMIENTAS DE SIMULACIÓN 14

3.1 N-CD++ 14
3.2 TRADUCTOR DE GEOMETRÍA DE CELDAS PARA MODELOS CELULARES BIDIMENSIONALES 16
3.2.1 Función de mapeo de grilla hexagonal a cuadrada 17
3.2.2 Función de mapeo de grilla triangular a cuadrada 18
3.2.3 Ltrans: un traductor de grillas con geometría triangular o hexagonal a cuadrada 18
3.3 *GRAFLOG* : UNA FORMA DE VER LOS RESULTADOS DE LAS SIMULACIONES DE MODELOS CELULARES BIDIMENSIONALES EN MODO GRÁFICO 20

4. APLICACIONES SIMPLES 21

4.1 MEDIO EXCITABLE 21
4.2 TENSIÓN SUPERFICIAL 23
4.3 DIFUSIÓN DE CALOR 25
4.4 COLISIÓN DE PARTÍCULAS UTILIZANDO LA VECINDAD DE MARGOLUS. 28
4.5 BUSCADOR DE CALOR 32
4.6 ROBOTS 36

5. APLICACIONES COMPLEJAS 41

5.1 HORMIGAS 41
5.2 WATERSHED 46
5.3 INCENDIOS FORESTALES 52
5.4 SOLIDIFICACIÓN BINARIA 57

6. CONCLUSIONES 62

7. BIBLIOGRAFÍA 62

8. BIBLIOGRAFÍA ADICIONAL 63

APÉNDICE 63

LENGUAJE DE ESPECIFICACIÓN DE N–CD++ 63
MANUAL DEL USUARIO DEL GRAFLOG 65

Tabla de Figuras

FIGURA 1 – FRAGMENTO DE UN AUTÓMATA CELULAR BIDIMENSIONAL....8
FIGURA 2: EVOLUCIÓN DEL JUEGO DE LA VIDA EN UN AC DE 100X100 CELDAS. DE IZQ. A DER. PODEMOS VER LA EVOLUCIÓN PARTIENDO DE UN ESTADO INICIAL CON VALORES ALEATORIOS....11
FIGURA 3 – REGLAS PARA EL *JUEGO DE LA VIDA*....14
FIGURA 4 - MAPEO DE GEOMETRÍA HEXAGONAL A CUADRADA....17
FIGURA 5 - DISTRIBUCIÓN DE LOS VECINOS CERCANOS EN EL MAPEO....17
FIGURA 6 - MAPEO DE GEOMETRÍA TRIANGULAR A CUADRADA....18
FIGURA 7 - DISTRIBUCIÓN DE LOS VECINOS CERCANOS EN EL MAPEO....18
FIGURA 8 - POSICIÓN RELATIVA DE LOS VECINOS EN LAS DISTINTAS GEOMETRÍAS....18
FIGURA 9 – EJEMPLO DE UNA REGLA EN UNA GEOMETRÍA HEXAGONAL CON SU CORRESPONDIENTE TRADUCCIÓN....19
FIGURA 10 – EJEMPLO DE UNA REGLA EN UNA GEOMETRÍA TRIANGULAR CON SU CORRESPONDIENTE TRADUCCIÓN....19
FIGURA 11(A) – RESULTADO OBTENIDO LUEGO DE CORRER EL DRAWLOG....20
FIGURA 11(B) – EL MISMO RESULTADO EN MODO GRÁFICO UTILIZANDO EL GRAFLOG....21
FIGURA 12 – IMPLEMENTACIÓN DE MEDIOS EXCITABLES....22
FIGURA 13 – RESULTADOS DE EXMEDIA CON DIFERENTES VECINDADES: (A) MOORE, (B) VON NEUMAN Y (C) GRILLA HEXAGONAL.....23
FIGURA 14 – IMPLEMENTACIÓN DEL MODELO DE TENSIÓN SUPERFICIAL....24
FIGURA 15 – RESULTADOS DE LA EJECUCIÓN DEL MODELO DE TENSIÓN SUPERFICIAL....25
FIGURA 16 – IMPLEMENTACIÓN DEL MODELO DE DIFUSIÓN DE CALOR....27
FIGURA 17 – RESULTADOS DE LA EJECUCIÓN DEL MODELO DE DIFUSIÓN DE CALOR....28
FIGURA 18 - REGLAS DE BLOQUES DE 2X2 CELDAS....29
FIGURA 19 - LOS BLOQUES DE 2X2 CELDAS CORRESPONDIENTES A LA GRILLA PAR (LÍNEAS GRUESAS) Y A LA GRILLA IMPAR (LÍNEAS FINAS). DEPENDIENDO DE CUAL GRILLA SE ESTE UTILIZANDO LA CELDA MARCADA EN (A) TENDRÁ COMO VECINOS A LOS QUE ESTÁN ALINEADAS CON LA GRILLA PAR (B) Y A LOS QUE ESTÁN ALINEADAS CON LA GRILLA IMPAR (C).....29
FIGURA 20 – REGLAS DEL MODELO HPP-GAS....30
FIGURA 21 – HPP-GAS: (A) MOVIMIENTO UNIFORME DE PARTÍCULAS AISLADAS. (B) PARTÍCULAS QUE COLISIONAN Y CAMBIAN DE DIRECCIÓN HACIA LA DIAGONAL OPUESTA.....30
FIGURA 22 – IMPLEMENTACIÓN DEL MODELO HPP-GAS....31
FIGURA 23 – (A) CELDA UBICADA EN UNA POSICIÓN CON FILA Y COLUMNA PAR ALINEADA CON LA GRILLA PAR. (B) CELDA UBICADA EN UNA POSICIÓN CON FILA Y COLUMNA IMPAR ALINEADA CON LA GRILLA IMPAR....32
FIGURA 24 – IMPLEMENTACIÓN DEL MODELO DEL BUSCADOR DE CALOR....35
FIGURA 25 – (A) EL MISIL SE DIRIGE HACIA EL ESTE LAS CELDAS QUE SE ENCUENTRAN EN SUS LATERALES DEFINIRÁN LA NUEVA DIRECCIÓN. (B) COMO LA CELDA QUE SE ENCONTRABA A LA DERECHA DEL MISIL TENIA MAYOR TEMPERATURA EL MISIL GIRÓ Y AVANZO HASTA ESA POSICIÓN TENIENDO AHORA UNA DIRECCIÓN HACIA EL NORTE. (C) SE MUESTRAN LAS NUEVAS CELDAS QUE DEFINIRÁN LA NUEVA DIRECCIÓN DEL MISIL.....36
FIGURA 26 – TRAZADO DE LAS RUTAS CON EL SENTIDO DE CIRCULACIÓN DE LOS ROBOTS.....38
FIGURA 27 – ESPECIFICACIÓN DEL MODELO ROBOTS.....40
FIGURA 28 – IMPLEMENTACIÓN DEL MODELO DE HORMIGAS....43
FIGURA 29 – VECINDAD EN EL MODELO HORMIGAS....43
FIGURA 30 – HORMIGA QUE PUEDE COLISIONAR CON HORMIGAS QUE VENGAN CON DIRECCIÓN OESTE O CON DIRECCIÓN ESTE....44
FIGURA 31 – EJECUCIÓN DEL MODELO HORMIGAS, LAS CELDAS BLANCAS REPRESENTAN LA AUSENCIA DE PASTO, LAS CELDAS AMARILLAS REPRESENTAN LA RAÍZ DEL PASTO, LAS CELDAS VERDES REPRESENTAN EL PASTO SIN COMER Y POR ULTIMO LAS CELDAS NEGRAS REPRESENTAN A LAS HORMIGAS. (A) LOS PRIMEROS CUATRO PASOS DONDE SE PUEDE VER COMO LA HORMIGA CAMBIA DE DIRECCIÓN CADA VEZ QUE COME PASTO. (B) LA HORMIGA AVANZA HASTA QUE ENCUENTRA PASTO NUEVAMENTE....45
FIGURA 32 – VISTA DESDE UN SISTEMA GIS DE UN WATERSHED....46
FIGURA 33 – MODELO HIDROLÓGICO CONCEPTUAL....46
FIGURA 34 – IMPLEMENTACIÓN DEL MODELO WATERSHED....48
FIGURA 35 – RESULTADOS DE LA EJECUCIÓN DEL MODELO WATERSHED. (A) TOPOLOGÍA DEL TERRENO. (B) LLUVIA DURANTE 5 MINUTOS. (C) LUEGO DE 10 MINUTOS. (D) LUEGO DE 15 MINUTOS. (E) LUEGO DE 20 MINUTOS. (F) LUEGO DE 30 MINUTOS. (G) LUEGO DE 45 MINUTOS EL TERRENO SE CUBRE EN SU TOTALIDAD TENIENDO UN NIVEL BASTANTE HOMOGÉNEO.....51
FIGURA 36 – FOTOGRAFÍAS DONDE SE MUESTRAN LA VEGETACIÓN EN LOS DIFERENTES MODELOS DE COMBUSTIBLES. (A) MODELO DE COMBUSTIBLE 1. (B) MODELO DE COMBUSTIBLE 2. (C) MODELO DE COMBUSTIBLE 3. (D) MODELO DE

COMBUSTIBLE 4. (E) MODELO DE COMBUSTIBLE 5. (F) MODELO DE COMBUSTIBLE 6. (G) MODELO DE COMBUSTIBLE 9. (H) MODELO DE COMBUSTIBLE 11. 54
FIGURA 37 – RESULTADO DE LA EJECUCIÓN DEL PROGRAMA QUE DETERMINA LA VELOCIDAD DE AVANCE DEL FUEGO. .55
FIGURA 38 – ESPECIFICACIÓN DEL MODELO DE FUEGO FORESTALES....56
FIGURA 39 – (A) AVANCE DEL FUEGO UTILIZANDO GRAFLOG. EL FUEGO SE ORIGINA EN LA CELDA (11,11) CON VIENTO EN DIRECCIÓN NORESTE. (B) GRÁFICO QUE REPRESENTA EL AVANCE DEL FUEGO A LO LARGO DE 2 HORAS. EL FUEGO SE ORIGINA EN LA ZONA DE COLOR MÁS OSCURO. CADA CAMBIO DE COLOR REPRESENTA 20 MINUTOS....57
FIGURA 40 – ESPECIFICACIÓN DEL MODELO DE SOLIDIFICACIÓN BINARIA....60
FIGURA 41 – ESTRUCTURA DE LOS CRISTALES....61

1. Resumen

En los últimos años, una gran variedad de sistemas complejos ha tomado un papel protagónico (ej. la computadora conectada una red de computadoras, controladores de tráfico aéreo, plantas industriales, aplicaciones embebidas, etc.). El costo de desarrollo de tales sistemas es crucial para una implementación exitosa, y su complejo análisis ha sido atacado usando simulaciones. El uso de una aproximación con modelos formales puede producir reducciones de costos importantes. A su vez la simulación permite experimentación controlada y compresión de tiempo (una simulación se realiza en mucho menos tiempo que el sistema real que modela). Otra gran ventaja es que su uso no afecta al sistema real, que puede seguir utilizándose (o no existir).

El objetivo de este trabajo consiste en la aplicación de paradigmas específicos para la simulación de diversos fenómenos físicos, biológicos, naturales y de comportamiento basados en eventos discretos.

Se presentara un conjunto importante de modelos simples y complejos. El trabajo está basado en una herramienta denominada N-CD++ y se ha organizado en siete secciones. Comienza con un resumen breve y objetivos del trabajo. Luego en la segunda sección planteo una introducción a la modelización y simulación de sistemas para luego introducir los formalismos DEVS, los Autómatas Celulares y Cell–DEVS. En la tercera sección se verá un análisis global de la implementación de la herramienta y las extensiones realizadas sobre la misma, junto con el lenguaje de especificación usado para la definición del comportamiento de las celdas del modelo celular. En la cuarta sección se describen los modelos simples y como fueron implementados. Una quinta sección describe modelos complejos y como fueron implementados. En la sexta sección se encuentran las conclusiones. Por ultimo la sección siete detalla la bibliografía utilizada para este trabajo.

2. Introducción

La simulación de un sistema requiere generalmente de los siguientes pasos:

1. Identificación y planteo del problema en función de objetos y actividades.
2. Análisis de los datos de entrada del sistema real y obtención de los mismos.
3. Construcción de un modelo del sistema con los aspectos que se quieren simular.
4. Implementación del modelo para poder ser ejecutado en una computadora.
5. Verificación y validación del modelo.
6. Ejecución de la simulación.
7. Análisis de los datos de salida de la simulación.

Se han desarrollado gran variedad de paradigmas de modelado, que pueden clasificarse de acuerdo a distintos criterios: con respecto a la *base de tiempo*, hay paradigmas a **tiempo continuo**, donde el tiempo evoluciona de forma continua, y a **tiempo discreto**, donde el tiempo avanza por saltos de un valor entero a otro. Con respecto a los conjuntos de *valores de las variables* descriptivas del modelo, hay paradigmas de estados o **eventos discretos** (las variables toman sus valores en un conjunto discreto), **continuos** (las variables son números reales), y **mixtos** (ambas posibilidades) **[Gia96]**.

En cuanto a la caracterización del problema a modelar, los modelos pueden ser **prescriptivos** si formulan y optimizan el problema (en general son métodos analíticos) o **descriptivos** si describen el comportamiento del sistema (suelen ser métodos numéricos).

Por otro lado, un modelo se dice que es **determinístico** si todas las variables tienen certeza completa y están determinadas por sus estados iniciales y sus entradas. El modelo se dice **probabilístico** si los cambios de estado se producen por medio de leyes aleatorias. Por ejemplo: los datos de entrada del modelo son aleatorios, siendo el modelo determinista, ó el tiempo de llegada de los eventos es aleatorio **[Zei96]**. Si usa variables aleatorias se dice que el modelo es **estocástico**.

Según el *entorno*, los modelos son **autónomos** (no existen entradas) o no autónomos. Los autónomos evolucionan únicamente en función de tiempo.

2.1 Formalismo DEVS

El formalismo DEVS (Discrete EVents dynamic Systems) propuesto por Bernard Zeigler **[Zei76,Zei84]** propone una teoría de modelado de sistemas a tiempo continuo usando modelado de eventos discretos.

Permite descripción modular de los fenómenos a modelar (aproximación modular) y ataca la complejidad usando una aproximación jerárquica.

El formalismo provee una forma de especificar un objeto matemático llamado sistema. Este se describe como un conjunto consistente de una base de tiempo, entradas, salidas y funciones para calcular los siguientes estados y salidas. Esta aproximación se integró posteriormente con nociones de programación orientada a objetos **[Zei84, Zei90, Zei95]**. Soporta la construcción de modelos de forma jerárquica y modular. Además de un medio para construir modelos simulables, provee una representación formal para manipular matemáticamente sistemas de eventos discretos.

Puede verse como una forma de especificar sistemas cuyas entradas, estados y salidas son constantes de a trozos, y cuyas transiciones se identifican como eventos discretos.

El formalismo define cómo generar nuevos valores para las variables y los momentos en los que estos valores deben cambiar. Los intervalos de tiempo entre ocurrencias son variables, lo que trae algunas ventajas: en los formalismos con una única granularidad es difícil describir los modelos donde hay muchos procesos operando en distintas escalas de tiempo, y la simulación no es eficiente ya que los estados deben actualizarse en el momento con menor incremento de tiempo, lo cual desperdicia tiempo cuando se aplica a los procesos más lentos.

Un modelo DEVS se construye sobre la base de un conjunto de modelos básicos llamados **atómicos**, que se combinan para formar modelos acoplados. Los modelos **atómicos** son objetos independientes modulares, con variables de estado y parámetros, funciones de transición internas, externas, de salida y avance de tiempo. Un modelo acoplado especifica cómo se conectan las entradas y salidas de los componentes.

Los nuevos modelos también son modelos modulares, y pueden usarse para armar modelos de mayor nivel.

El modelado jerárquico permite la creación de una Base de Datos de modelos, permitiendo la reutilización de los modelos creados y chequeados. En esta forma mejora la seguridad de la simulación, reduciendo el tiempo de chequeo de las nuevas simulaciones, y mejorando la productividad.

2.1.1 Modelos Atómicos

Un modelo atómico DEVS puede describirse formalmente como:

$$M = < X, Y, I, S, \delta_{INT}, \delta_{EXT}, \lambda, D >$$

donde:

X = conjunto de eventos externos de entrada

Y = conjunto de eventos externos de salida

$I = < P^X, P^Y >$, representa la interfaz del modelo. En este caso, $\forall\ i \in \{X, Y\}$, P_j^i es una definición de un puerto (de entrada o salida respectivamente), donde $j \in \mathbf{N}$, $j \in [1, \mu]$, ($\mu \in \mathbf{N}$, $\mu < \infty$), y

P_j^i = { (N_j^i, T_j^i) / $N_j^i \in [i_1, i_\mu]$ (nombre del port), y T_j^i = Tipo del port}

S = conjunto de variables de estados y parámetros. En general se usan las variables *phase* y *sigma* para representar el estado del modelo y el tiempo restante para el próximo cambio de estado

δ_{INT}: S $\rightarrow$ S, es la función de transición interna, que especifica el cambio de estado por causa de eventos internos;

δ_{EXT}: QxX $\rightarrow$ S, es la función de transición externa, que especifica los cambios de estado por causa de eventos externos, y posiblemente, una replanificación en su próxima transición interna. Q es el conjunto de estados totales del sistema especificado como Q = { (s,e) / s $\in$ S, e $\in$ [0, D(s)] }, donde **e** representa el tiempo transcurrido desde la última transición de estado con estado **s**;

λ: S $\rightarrow$ Y, es la función de salida, que genera los resultados en los puertos con anterioridad a la ejecución de la función de transición interna;

D: S $\rightarrow$ $\boldsymbol{R}_0^+ \cup \infty$, función de avance de tiempo, que controla la frecuencia de las transiciones internas. D(s) es el tiempo que el modelo se queda en el estado s si no hay un evento externo.

Los modelos poseen puertos de entrada y salida a través de los cuales interactúan con el entorno. Los eventos determinan los valores que aparecen en esos puertos. Los datos externos son recibidos en un puerto de entrada y la especificación del modelo debe determinar cómo responder a dichos eventos. Los eventos internos que se producen dentro del modelo, modifican su estado y producen eventos destinados a los puertos de salida. Las influencias de los puertos de salida determinarán si estos datos serán enviados a otros componentes como eventos externos.

2.1.2 Modelos Acoplados

Un modelo atómico puede integrarse con otros modelos DEVS para formar un modelo acoplado. Incluso, varios modelos acoplados pueden formar un nuevo modelo acoplado. Estos se definen como:

$$C = < X, Y, I, D, \{ M_i \}, \{ I_i \}, \{ Z_{ij} \}, select >$$

donde:

X = conjunto de eventos externos de entrada

Y = conjunto de eventos externos de salida

I = < P^X, P^Y>, representa la interfaz del modelo. En este caso, $\forall\, i \in$ {X, Y}, P^i es una definición de un puerto (de entrada o salida respectivamente), donde:

P_j^i = { (N_j^i, T_j^i) / $\forall$ j $\in$[1, μ], ($\mu \in \boldsymbol{N}$, $\mu < \infty$), $N_j^i \in [i_1, i_\mu]$ (nombre del port), y T_j^i = Tipo del port}

D = conjunto de nombres de los componentes que lo conforman
M_i = modelo básico correspondiente al componente i, definido como:

$$M = < X_i, Y_i, I_i, S_i, \delta_{INT,i}, \delta_{EXT,i}, \lambda_i, D_i >$$

$I_i \subseteq D$, es el conjunto de modelos influenciados por el modelo i

$Z_{ij} = Y_i \rightarrow X_j$, es la función de traducción de la influencia *i* en *j*

select: D $\rightarrow$ D, es la función para determinar prioridades ante eventos simultáneos.

2.2 Autómatas Celulares

Los autómatas celulares son un formalismo para modelado de sistemas dinámicos complejos, de variables y tiempo discreto (el tiempo, espacio y estados del sistema son discretos), creados originalmente por von Neumann y S. Ulam. Un autómata celular es un conjunto infinito n-dimensional de celdas ubicadas geométricamente**[Tof87]**. Cada punto de la grilla espacial infinita (llamado una celda) puede tener uno de un conjunto de valores finitos de estados, es decir que cada celda tiene un estado elegido de un alfabeto finito. Cada una contiene el mismo aparato de cálculo que las otras y se conectan entre sí de forma uniforme. El estado de las celdas se actualiza simultáneamente y de forma independiente de los demás en pasos de tiempo discreto. Para ello se define la vecindad de una celda, que es un conjunto de celdas cercanas. Esta vecindad es homogénea para todas las celdas, como así también la geometría (cuadrada, hexagonal y triangular)

Los estados de las celdas en la grilla se actualizan de acuerdo con una regla local. O sea, el estado de una celda en un momento dado sólo depende de su propio estado en el instante de tiempo previo, y los estados de sus vecinos en ese mismo momento. Todas las celdas de la grilla se actualizan sincrónicamente.

Luego, el estado de toda la celda avanza en pasos de tiempo discreto. Las celdas actualizan sus valores usando una función de transición, que toma como entrada el estado actual de la celda, y un conjunto finito de celdas cercanas, que están a una distancia limitada (la vecindad). La uniformidad en el espacio significa que el sistema posee invariancia espacial y en tiempo.

Pueden usarse para modelar variedad de sistemas discretos naturales, pero también sirven para comprender muchos formalismos comunes que pueden verse como extensiones de concepto básico. Se suelen usar, al igual que las ecuaciones diferenciales para el modelado de sistemas físicos. Existen autómatas para modelar dinámica en fluidos y gases, colonias de animales, modelos ecológicos, entre otros. También tienen aplicación en criptografía. Permiten especificar modelos de sistemas complejos con distintos niveles de descripción, por lo que permiten atacar mayor complejidad que las ecuaciones diferenciales, permitiendo el modelado y estudio de sistemas muy complejos.

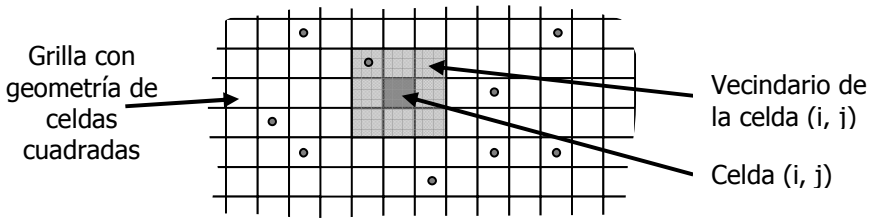


Figura 1 – Fragmento de un Autómata Celular Bidimensional

Formalmente, un autómata celular se define como:

$$CA = < A, S, n, \{t_1,...,t_n\}, C, \eta, N, B, T, \tau, c.Z_0^+ >$$

donde:

A = $\{A \subseteq \boldsymbol{R} \wedge \#A < \infty\}$: Alfabeto de estados de celdas

S = $\{S \in A\}$: Conjunto de posibles estados para cada celda

n $\in$ ***N***, es la dimensión del espacio de celdas

$\{\mathbf{t_1}, ..., \mathbf{t_n}\}$ es la cantidad de celdas en cada una de las dimensiones

C = $\{ C_C / c = (i_1,...,i_n)$, con $0 \le i_k \le t_k \ \forall \ k = 1...n$, es la posición de la celda c en el espacio n-dimensional $\wedge \ C_C \in S \}$, es el conjunto de estados de cada una de las celdas del autómata

η es la cantidad de celdas que integran el vecindario

N = { $(v_{k1},...,v_{kn})$, $\forall$ k = 1... η } es la definición del vecindario como un desplazamiento

B es el conjunto de celdas de borde. Se define:

B = {$\varnothing$} si el espacio de celdas es toroidal; o
B = {C_b / $C_b \in C$, con $b \in L$}, siendo L = { $(i_1,...,i_n)$ / $i_j = 0 \vee i_j = t_j \; \forall \; j \in [1, n]$ }, y sujeto a que $\tau_b \neq \tau_c = \tau \; \forall \; c \notin L$.

T = {C x $c.Z_0^+ \rightarrow$ C}: Función de transición global

τ = {C_i x N x $c.Z_0^+ \rightarrow C_k$}: función de cómputo local

$c.Z_0^+$: Base de tiempo (discreto) del autómata celular

$c.Z_0^+$ = { i / $i \in N$, i = c.j donde $j \in N$} = { 0, 1c, 2c, 3c,...}

El formalismo utiliza una base de tiempo discreta, la cual restringe la precisión y eficiencia de los modelos simulados. En el caso particular de autómatas celulares que contienen un número considerable de celdas y reglas de alto grado de complejidad para ser resueltas, será necesaria una gran cantidad de tiempo de cómputo para obtener el grado de precisión deseado. Más allá de esto, en muchos casos la mayoría de las celdas del autómata no necesitan actualización. La presencia de celdas inactivas permite la definición de un nuevo paradigma llamado **autómatas celulares asincrónicos**.

Se llama un evento en el autómata a un cambio de estado en una celda. Por ende, se puede decir que en un autómata celular los eventos pueden ocurrir en un número reducido de celdas (aunque es posible que ocurra en todas simultáneamente), y que puede detectarse si ocurrirá un evento en una celda mirando si hubo eventos en sus vecinos. En este caso los eventos pueden ocurrir en un instante impredecible de tiempo por lo cual se aproxima a una base de tiempo continuo, lo que permite lograr mayor precisión y evitar la simulación de los períodos de inactividad de las celdas, mejorando así la utilización de los recursos computacionales y obteniendo una simulación con mejor rendimiento. Sin embargo, este enfoque requiere la sincronización explícita de las celdas, incrementando así la complejidad de las rutinas y algoritmos que llevan a cabo la simulación. Por lo tanto se produce una sobrecarga inherente al problema mencionado con anterioridad. En algunos casos esta sobrecarga puede anular las mejoras obtenidas con el enfoque asincrónico.

Para lograr simular estos modelos, se debe mantener una lista de últimos eventos, que contiene las celdas cuyos eventos ocurrieron recientemente, junto con la hora de ocurrencia de los mismos. Luego de simular cada una de las transiciones, se chequea si hubo un cambio de estado, y si es así la celda se agrega en la lista de últimos eventos. Este mecanismo puede causar distintos resultados dependiendo del orden de ejecución de las celdas, especialmente cuando las mismas son evaluadas en paralelo, transformando al autómata en un modelo no determinístico. Para evitar estos problemas, todas las celdas activas que figuren en la lista de eventos deben ejecutar sobre el mismo espacio de celdas de base, antes de hacerle cualquier modificación.

Formalmente un Autómata Celular Asincrónico ejecutable se define como:

ACA = < A, S, n, $\{t_1,...,t_n\}$, C, η, N, B, Nevs, T, τ, $\boldsymbol{R}_0^+$>

donde:

A = {$A \subseteq \boldsymbol{R} \wedge \#A < \infty$}: Alfabeto de estados de celdas

S = {S ∈ A}: Conjunto de posibles estados para cada celda

n ∈ ***N***, es la dimensión del espacio de celdas

{$\mathbf{t_1}$, ..., $\mathbf{t_n}$} es la cantidad de celdas en cada una de las dimensiones

C = { C_C / c = (i_1,...,i_n), con $0 \le i_k \le t_k \ \forall$ k = 1...n, es la posición de la celda c en el espacio n-dimensional $\wedge$ $C_C \in$ S }, es el conjunto de estados de cada una de las celdas del autómata

η es la cantidad de celdas que integran el vecindario

N = { (v_{k1},...,v_{kn}), ∀ k = 1... **η** } es la definición del vecindario como un desplazamiento

B es el conjunto de celdas de borde. Se define:

B = {∅} si el espacio de celdas es toroidal; o
B = {C_b / $C_b \in$ C, con b ∈ L}, siendo L = { (i_1,...,i_n) / $i_j = 0 \vee i_j = t_j \ \forall$ j ∈ [1, n] }, y sujeto a que $\tau_b \neq \tau_c = \tau \ \forall$ c ∉ L.

Nevs = { (c, *t*) / c ∈ C $\wedge$ *t* ∈ $\boldsymbol{R}_0^+$}, es una lista de próximos eventos, donde *c* es la posición de una celda en el espacio, y *t* es la hora del evento correspondiente

T = {C x $\boldsymbol{R}_0^+ \rightarrow$ C}: Función de transición global

τ = {C_i x N x $\boldsymbol{R}_0^+ \rightarrow C_k$}: función de cómputo local

$\boldsymbol{R}_0^+$: Base de tiempo (continua) del autómata celular

Cada celda en el espacio puede tomar un valor del conjunto *S*. La evolución del autómata se define por la ejecución de la función de transición global (**T**) que actualiza el estado en todo el espacio de celdas (aunque en los autómatas celulares asincrónicos esta función actúa solo sobre las celdas activas). El comportamiento de esta función de transición depende de los resultados de la función de cómputo local (**τ**), que se ejecuta localmente en cada vecindario perteneciente a una celda (**N**). Conceptualmente, el cómputo de esta función de transición local se realiza en forma sincrónica y paralela en cada celda del modelo.

Para cada autómata celular bidimensional la función de transición local en el instante de tiempo *t* se define como:

$$s_{ij}[t+1] = \tau(s_{i_0}, j_0[t], \ldots, s_{i_\eta}, j_\eta[t])$$

Aquí $s_{i,j}[t]$ representa el estado de la celda (i, j) en un tiempo de simulación *t*, y τ denota la función de cálculo local. Los parámetros s_{i_0}, j_0,..., s_{i_η}, j_η definen el vecindario de la celda. Una vez hecha la definición para dos dimensiones esto puede extenderse a n.

Un ejemplo muy conocido es el **"Juego de la Vida"**. Este AC fue desarrollado por el matemático John Conway y popularizado por Martin Gardener en su columna de los Juegos Matemáticos de la revista Scientific American en octubre 1970. El juego de la vida es uno de los ejemplos más simples de un AC. Cada celda o está viva (1) o muerta (0). Para determinar el estado de una celda en el próximo paso de tiempo, se cuenta el número de celdas vecinas que están vivas. Si está viva y tiene 2 vecinos vivos o 3 vecinos vivos, entonces la celda permanecerá viva durante el próximo paso de tiempo. Menos vecinos vivos se muere de soledad, más vecinos hace que se muera de superpoblación.

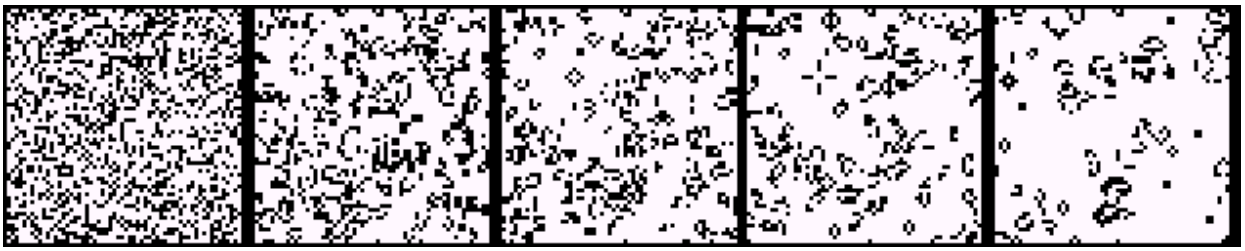


Figura 2: Evolución del juego de la vida en un AC de 100x100 celdas. De izq. a der. podemos ver la evolución partiendo de un estado inicial con valores aleatorios

En la Figura 2 se puede apreciar la evolución del autómata celular a lo largo del tiempo. De izquierda a derecha se aprecia como, partiendo de una distribución aleatoria de partículas, con el paso del tiempo van desapareciendo algunas debido a que se mueren por superpoblación. En la ultima vista se puede apreciar al modelo, ya mucho más estable, con determinados patrones que se repiten a lo largo de la superficie.

2.3 Formalismo Cell–DEVS

Cell-DEVS **[Wai98]** es un formalismo que se basa en DEVS, extendiéndolo para poder implementar autómatas celulares. En el paradigma Cell-DEVS cada celda se define como un modelo atómico con un conjunto de estados para los valores del vecindario, un comportamiento y una demora de actualización que puede ser de transporte ó inercial **[GM76]**. Un modelo acoplado que incluya a un conjunto de estas celdas formará así a un modelo celular.

Los modelos atómicos Cell-DEVS con dominio en los reales, pueden describirse formalmente como:

$$CA = < X, Y, I, demora, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

dónde para el alfabeto A, con $\#A < \infty \wedge A \subseteq \boldsymbol{R} \cup \{ T, F, ? \}$:

$\boldsymbol{X} \subseteq A$ es el conjunto de eventos externos de entrada

$\boldsymbol{Y} \subseteq A$ es el conjunto de eventos externos de salida

$\mathbf{I} = < \eta, \mu, P^X, P^Y >$ representa la definición de la interfaz del modelo. En este caso,

$\eta \in \boldsymbol{N}$, $\eta < \infty$ es el tamaño de la vecindad,

$\mu \in \boldsymbol{N}$, $\eta < \infty$ es la cantidad de puertos de entrada/salida independientes de la vecindad,

P^X es el puerto de entrada que acepta valores de A.

P^Y es el puerto de salida que acepta valores de A.

$\forall j \in [1, \eta]$, $i \in \{X, Y\}$ definimos el puerto: $P_j^i = \{ (N_j^i, T_j^i) /$

$\forall j \in [1, \eta+\mu]$, $N_j^i \in [i_1, i_{\eta+\mu}]$ (nombre del puerto), y $T_j^i \in I_i$ (Tipo del puerto)$\}$, $I_i = \{ x / x \in X \text{ si } i = X \}$ ó $I_i = \{ x / x \in Y \text{ si } i = Y \}$

La **demora** puede ser *de transporte* o *inercial*

$\mathbf{S} \subseteq A$ incluye todos los valores posibles de estados para la celda

θ es la definición del estado de la celda, definido como:

- Si la demora es de <u>transporte</u>:

  $\theta = \{ (s, phase, \sigma_{queue}, \sigma) /$

$s \in S$ es el valor del estado para la celda
phase $\in$ {activa, pasiva}
$\sigma_{queue} = \{ ((v_1,\sigma_1),...,(v_m,\sigma_m)) / m \in \boldsymbol{N} \wedge m < \infty) \wedge \forall (i \in \boldsymbol{N}, i \in [1,m]), v_i \in S \wedge \sigma_i \in \boldsymbol{R}_0^+ \cup \infty\}$

$\sigma \in \{\boldsymbol{R}_0^+ \cup \infty\}$

- Si la demora es <u>inercial</u>:

$\theta = \{ (s, phase, \sigma) /$

$s \in S$ es el valor del estado para la celda
phase $\in$ {activa, pasiva}
$\sigma \in \{\boldsymbol{R}_0^+ \cup \infty\}$

N $\in S^{\eta+\mu}$ es el conjunto de estados de los eventos de entrada almacenados

d $\in \boldsymbol{R}_0^+$, $d < \infty$ es la demora de la celda

$\boldsymbol{\delta_{int}}$: $\theta \rightarrow \theta$ es la función de transición interna

$\boldsymbol{\delta_{ext}}$: $Qx\boldsymbol{X} \rightarrow \theta$ es la función de transición externa, donde Q es el conjunto definido como:

$Q = \{ (s, e) / s \in \theta \times N \times d; e \in [0, D(s)]\}$

$\boldsymbol{\tau}$: $N \rightarrow S$ es la función de cálculo local;

$\boldsymbol{\lambda}$: $S \rightarrow \boldsymbol{Y}$ es la función de salida; y

D: $\theta \times N \times d \rightarrow \boldsymbol{R}_0^+ \cup \infty$, es la función de duración de vida del estado.

Aquí **D**(s, phase, σ_{queue}, σ, N, d) = *t* representa el tiempo durante el cual, si no hay eventos externos, el modelo atómico conservará el estado actual.

La especificación permite definir una celda como un modelo atómico modular. Un modelo Cell-DEVS acoplado n-dimensional puede definirse como:

$$CC = < Xlist, Ylist, I, \boldsymbol{X}, \boldsymbol{Y}, n, \{t_1,...,t_n\}, \eta, N, C, B, Z, select >$$

donde:

Ylist = $\{ (k_1, k_2, ..., k_n) / k_i \in [0,t_i] \forall i \in [1,n], i \in \boldsymbol{N}\}$, es la lista de acoplamiento externo

Xlist = $\{ (k_1, k_2, ..., k_n) / k_i \in [0,t_i] \forall i \in [1,n] , i \in \boldsymbol{N}\}$, es la lista de acoplamiento interno

I = $< P^X, P^Y >$, es la interfaz de comunicación con los modelos externos, donde:
P^x es un port de entrada que acepta valores de $\boldsymbol{R}$.
P^y es un port de salida que acepta valores de $\boldsymbol{R}$.

n $\in \boldsymbol{N}$, $n < \infty$ es la dimensión del espacio de celdas

$\boldsymbol{\{t_1,..., t_n\}}$, con $t_i \in \boldsymbol{N}$, $1 \leq i \leq n$, es la cantidad de celdas en cada una de las dimensiones

$\boldsymbol{\eta} \in \boldsymbol{N}$, es el tamaño del vecindario;

$\boldsymbol{X} \subseteq \boldsymbol{R}$, es el conjunto de eventos externos de entrada;

$\boldsymbol{Y} \subseteq \boldsymbol{R}$, es el conjunto de eventos externos de salida;

N = { $(i_{1,p}, i_{2,p}, ..., i_{n,p})$ / $i_{j,p} \in \boldsymbol{Z} \forall j \in \boldsymbol{N}, j \in [1,n], \forall p \in \boldsymbol{N}, p \in [1,\eta]$ }. Es la definición del vecindario, expresada como un desplazamiento con respecto a la celda central.

C = $\{C_{k1, k2, ..., kn} / k_i \in [0,t_i] \forall i \in \boldsymbol{N}, i \in [1,n]\}$, es el conjunto de celdas atómicas que conforman al modelo acoplado, donde:

$C_{k1, k2, ..., kn} = < X_{k1, k2, ..., kn}, Y_{k1, k2, ..., kn}, I_{k1, k2, ..., kn}, demora_{k1, k2, ..., kn}, S_{k1, k2, ..., kn}, \theta_{k1, k2, ..., kn}, N_{k1, k2, ..., kn}, d_{k1, k2, ..., kn}, \delta_{int, k1, k2, ..., kn}, \delta_{ext, k1, k2, ..., kn}, \tau_{k1, k2, ..., kn}, \lambda_{k1, k2, ..., kn}, D_{k1, k2, ..., kn} >$

B es el conjunto de celdas que representan el borde del modelo celular, donde:

- B = {∅} si el espacio de celdas es toroidal (Wrapped) o
- B = $\{C_{k1, k2, ..., kn} / (k_1 = 0 \vee k_1 = t_1) \wedge (k_2 = 0 \vee k_2 = t_2) \wedge ... \wedge (k_n = 0 \vee k_n = t_n) \wedge C_{k1, k2, ..., kn} \in C \wedge C_{k1, k2, ..., kn} = < X_{k1, k2, ..., kn}, Y_{k1, k2, ..., kn}, I_{k1, k2, ..., kn}, demora_{k1, k2, ..., kn}, S_{k1, k2, ..., kn}, \theta_{k1, k2, ..., kn}, N_{k1, k2, ..., kn}, d_{k1, k2, ..., kn}, \delta_{int, k1, k2, ..., kn}, \delta_{ext, k1, k2, ..., kn}, \tau_{k1, k2, ..., kn}, \lambda_{k1, k2, ..., kn}, D_{k1, k2, ..., kn} >$ es un componente atómico }, si las celdas atómicas del borde tienen distinto comportamiento que el resto del espacio de celdas

Z: $I_{k1, k2, ..., kn} \rightarrow I_{w1,w2,...,wn}$, define el acoplamiento interno (conexión de puertos de E/S entre celdas):

$Z(P_{k1, k2, ..., kn}{}^{Y_q}) = P_{w1,w2,...,wn}{}^{X_q}$, con $(q \in \boldsymbol{N}, q \in [1,\eta]) \wedge \forall (r_1,r_2,...,r_n) \in N$, $w_1 = (k_1+r_1) \bmod t_1$; $w_2 = (k_2+r_2) \bmod t_2$; ...; $w_n = (k_n+r_n) \bmod t_n$ y

$Z(P_{k1, k2, ..., kn}{}^{X_q}) = P_{w1,w2,...,wn}{}^{Y_q}$, con $(q \in \boldsymbol{N}, q \in [1,\eta]) \wedge \forall (s_1,s_2,...,s_n) \in N$, $w_1 = (k_1-s_1) \bmod t_1$; $w_2 = (k_2-s_2) \bmod t_2$; ...; $w_n = (k_n-s_n) \bmod t_n$

select = { $(k_1, k_2, ..., k_n)$ / $k_i \in [0,t_i] \forall i \in \boldsymbol{N}, i \in [1,n]$ } es la función de selección de una celda inminente ante eventos simultáneos.

Las celdas con **demora de transporte** usan una cola para mantener los valores de los resultados de los cálculos junto con su hora futura de planificación, ya que durante la demora pueden llegar nuevos eventos externos. Cuando llega un evento por medio de la función de transición externa, la función de cómputo local es invocada, la cual, utilizando los valores de todas las entradas disponibles (el vecindario y los valores ingresados por los puertos) obtiene como resultado un estado al cual la celda debe cambiar, y una demora. Si este estado es distinto al anterior debe encolarse en la cola de próximos eventos **[WFG97,WFG97b]**. Debido a que una celda es independiente de la otra, cada una guarda una copia de los estados de los vecinos, así como el estado anterior.
Al arribar un evento interno se invocan primero la función de salida, que envía como resultado al primer valor encolado, y luego la función de transición interna, que elimina el primer elemento de la cola y luego analiza si la cola está vacía. Si existe un elemento significa que el modelo debe reprogramarse en función de lo encolado, en caso contrario debe pasivarse y por lo tanto su próximo cambio de estado será a la hora infinito **[BBW98]**. La celda se mantiene activa mientras haya eventos en la cola, cuando ésta se vacía la celda se pasivará retornando como hora de próximo evento el valor infinito.

Por otro lado existen las celdas con **demora inercial**. Las mismas son útiles cuando se desea representar una semántica con remoción para el comportamiento de una celda. Frente a un arribo de un evento externo la celda ejecuta la función de cómputo local y obtiene como resultado un estado y una demora. Si el nuevo valor obtenido difiere del valor anterior la celda analiza tiempo restante para el

próximo evento interno. Si no existe planificación alguna la celda se programa con la demora recién calculada. En caso de existir una programación se analiza el valor de σ (tiempo restante) para comprobar si es mayor que cero, de ser así se descarta la programación anterior y se toma la nueva. De no ser así, continúa con lo planificado. Frente a un evento interno la celda realiza la función de salida con el valor calculado, y en la función de transición interna cambia su estado a pasivo, ya que no tiene más eventos para programar **[BBW98]**.

3. Herramientas de simulación

A continuación se detallan las herramientas utilizadas para simular los modelos de estudio. En primer lugar se tiene una breve introducción sobre el motor de simulación denominado **N-CD++**. A continuación se describen las dos herramientas desarrolladas para complementar al motor de simulación: el traductor de geometría de celdas y el graficador de resultados.

3.1 N-CD++

N–CD++ [RW99] es una extensión de la herramienta CD++ que permite la creación de modelos Cell–DEVS n–dimensionales, donde el estado de cada celda pertenece al conjunto ***R*** ∪ { **?** } donde **?** representa al valor *indefinido*. Incluye un lenguaje de especificación que permite describir el comportamiento de cada celda de un modelo celular cuyo BNF se encuentra en el apéndice. Además, permite definir el tamaño del espacio celular y su conexión con otros modelos DEVS (si los hubiese), así como también el tipo de demora, el vecindario, el borde y el estado inicial de cada celda. Para ello se siguen las definiciones teóricas del formalismo Cell-DEVS.

La especificación del comportamiento de una celda se logra definiendo un conjunto de reglas de la forma:

VALOR DEMORA { CONDICIÓN }

Cada regla indica que si se satisface la condición, el estado de la celda debe cambiar por el valor designado. Luego, debe demorarse usando el tiempo especificado. Si la condición no es verdadera, entonces se evaluará la siguiente regla (secuencialmente según el orden en que fueron definidas). Este proceso se repetirá hasta que una regla sea satisfecha (considerando sólo la primera que lo haga), o hasta que no haya más reglas. En este último caso, se producirá un error abortando la simulación. La ocurrencia de este error indica que el modelo ha sido especificado en forma incompleta. Existen otros errores que pueden abortar la ejecución, y generalmente se deben a la falta o a una incorrecta declaración de ciertos parámetros que describen las características del modelo (como la dimensión del espacio celular, el tipo de demora, la definición del vecindario, o los valores iniciales a ser usados por el modelo), o a la definición de un acoplamiento entre modelos que involucre un puerto inexistente.

Un conjunto de reglas permite definir el comportamiento para las celdas de un modelo celular. Si se considera, por ejemplo, el comportamiento del **"Juego de la Vida"**, cuyas reglas se muestran en la Figura 3. Esta especificación indica que si la celda esta activa y la cantidad de vecinos activos (incluyendo la celda en cuestión) es 3 ó 4, entonces la celda seguirá activa (con una demora de transporte de 10 milésimas de segundo). Si la celda esta inactiva, y el vecindario tiene 3 elementos activos, entonces la celda pasa a estar activa. Sino, la celda pasa a estar inactiva.

```
Rule: 1 10 { (0,0) = 1 and ( truecount = 3 or truecount = 4 ) }
Rule: 1 10 { (0,0) = 0 and truecount = 3 }
Rule: 0 10 { t }
```

Figura 3 – Reglas para el *Juego de la Vida*

El formalismo para Autómatas Celulares ha sido extendido por diversidad de autores, permitiendo incluir distintas propiedades. Entre ellas se destacan:

- **Autonomía**: un modelo celular puede tener entradas externas, independientemente de su vecindario. En **N-CD++**, tales entradas se corresponden a puertos que conectan un modelo DEVS con una celda específica del modelo celular.

- **Homogeneidad**: cada celda del modelo puede tener distintas reglas y conexiones, siendo en estos casos un autómata celular inhomogéneo. En **N-CD++** esto se logra mediante la definición de *zonas*.

- **Uniformidad:** otras extensiones permiten que los vecinos no sean las celdas más cercanas, permitiendo el uso de vecindarios más extensos.

- **Computabilidad**: para que el modelo celular pueda ser simulado, el mismo debe acotarse a un numero finito de celdas en cada paso. La forma más simple de hacerlo es limitando el modelo a un área finita, lo que hace que se pierda homogeneidad, ya que debe determinarse qué se hace con los bordes. Para esto suelen usarse dos aproximaciones: o los estados de los bordes se especifican desde afuera, o se conectan los extremos entre sí, implementando un autómata toroidal.

- **Determinismo:** un autómata celular asincrónico estocástico puede obtenerse definiendo un experimento aleatorio e incluyendo variables aleatorias en las reglas que definen el comportamiento del modelo. Para ello, la función de transición local debe permitir el uso de variables aleatorias.

Todas estas propiedades son soportadas por **N-CD++**. A su vez **N-CD++** soporta un conjunto de posibles valores para un estado, permitiendo así que el estado de una celda tenga un valor perteneciente al conjunto $\boldsymbol{R} \cup \{ \textbf{?} \}$. De aquí en más al citar el término **número real** se estará referenciando a un elemento de este conjunto.

Debido a errores de representación en la computadora, como así también a errores de redondeo, es posible que dos valores numéricos que teóricamente deberían ser iguales, no lo sean. Para solucionar este inconveniente, se permite definir que dos valores *a* y *b* sean iguales si *a* pertenece al intervalo $[b-\Delta, b+\Delta]$. El valor de Δ puede ser especificado por el modelador a través de un parámetro en la invocación al simulador, variando así la precisión considerada al efectuar los cálculos.

N-CD++ dispone además de un preprocesador que proporciona ciertas facilidades al lenguaje y que actúa sobre el archivo de definición de modelos, antes de la carga de los mismos. Dicho preprocesador efectúa las tareas de expansión de macros y descarte de comentarios. El uso de macros permite la escritura de funciones de transición, reglas y/o definición de modelos o parte de ellos, en archivos independientes, facilitando su reusabilidad.

El lenguaje para la definición de reglas que dan comportamiento a las celdas, posee funciones básicas para manipular los valores de la lógica trivalente (True, False y Indefinido).

El lenguaje dispone también de las operaciones aritméticas básicas (+, -, * y /). Además, se tienen distintos tipos de funciones para operar sobre **números reales**, como por ejemplo, funciones trigonométricas, obtención de raíces, potencias, redondeo y truncamiento de valores reales a enteros, generación de números primos, módulo, logaritmos, valor absoluto, factoriales, como así también cálculo de mínimos, máximos, M.C.D. y M.C.M; considerando que cualquier expresión o función que incluya al valor **?** (indefinido) dará como resultado **?**. Otras funciones existentes permiten chequear si un **número real** es entero, si es par o impar, si es un número primo, ó si es un valor indefinido.

Existen funciones para obtener la cantidad de celdas del vecindario cuyo estado tiene cierto valor. Por ejemplo, **truecount** devuelve la cantidad de celdas en estado 1. También están disponibles las funciones **falsecount**, **undefcount** y **statecount(n)**. Esta última es la más genérica y permite especificar el valor del estado a contabilizar.

También se cuenta con la función **cellPos**, que devuelve la i–ésima posición dentro de la tupla que referencia a la celda que esta ejecutando la función de transición, es decir, dada la celda (x_0,x_1,...,x_n), entonces cellPos(i) = x_i.

El lenguaje provee el uso de constantes predefinidas. Entre ellas se encuentran: pi y e, junto a ciertas constantes de uso frecuente en los dominios de la física y la química (como la constante gravitacional, de aceleración, velocidad de la luz, constante de Planck, etc.). También se definió la constante INF, que representa al valor infinito. Esta última se devuelve automáticamente cuando la evaluación de una expresión numérica produce un desborde numérico.

Existen también funciones para obtener valores dependiendo del resultado de la evaluación de cierta condición, tal es el caso de la función **IFU(c, t, f, u)** que devuelve t si la condición c evalúa a Verdadero; f si evalúa a Falso; y u si evalúa a Indefinido; y la función **IF(c, t, f)** que devuelve t si c evalúa a Verdadero, y f en otro caso.

Por otro lado se cuentan con funciones para la conversión de valores entre distintas unidades, las funciones **RadToDeg** y **DegToRad** se usan para la conversión de ángulos expresados en radianes a grados sexagesimales y viceversa, respectivamente. También se cuentan con funciones para la conversión de coordenadas polares a rectangulares y viceversa, y para la conversión de valores que representan temperaturas en grados Centígrados, Fahrenheit o Kelvin.

Finalmente se cuentan con funciones para la generación de números aleatorios, usando distintas distribuciones (Uniforme, Chi Cuadrado, Beta, Exponencial, F, Gama, Normal, Binomial y Poisson), lo que permite la simulación de modelos estocásticos.

Si se utilizan funciones de generación de números aleatorios en la definición de la condición de la regla, es posible que su evaluación dé como resultado el valor Falso. Si se considera por ejemplo la regla:

10 100 { random >= 0.4 }

cuya condición es evaluada a Verdadera en el 60 % de los casos, y a Falsa en el resto. Esto puede provocar que en algunos casos, todas las reglas sean evaluadas a Falso, mientras que en otros casos exista una o más que no lo hagan. En este caso, por tratarse de un modelo estocástico, la herramienta automáticamente identifica esta situación y si hay más de una regla que es evaluada a Verdadero, se considera sólo la primera que lo haga, mientras que si todas las evaluaciones son Falsas, se asigna el valor **?** a la celda, y se utiliza el tiempo de demora establecido por defecto en la definición del modelo, informando tal situación al modelador y prosiguiendo con la simulación.

3.2 Traductor de geometría de celdas para modelos celulares bidimensionales

Al conjunto homogéneo y regular de celdas en un autómata celular se lo denomina grilla. Los tipos de geometría de grillas mas utilizados son: triangular, cuadrada y hexagonal. Las principales características de cada geometría son:

- **Grilla Triangular**: La ventaja de este tipo de grilla es que cada celda tiene un numero reducido de vecinos cercanos (tres) lo cual en algunos modelos es muy necesario. Por otro lado las desventajas son la dificultad de representación y visualización.

- **Grilla Cuadrada**: Las ventajas son su simple representación usando arreglos de celdas y a su vez la simple visualización. La desventaja es que tiene insuficientes **isotropía**.

- **Grilla Hexagonal**: La ventaja es que es la geometría con mas alta **isotropía** de las tres, esto hace que las simulaciones sean mas naturales y en algunos casos es absolutamente necesario para modelar ciertos fenómenos. La desventaja es que es muy difícil de representar y visualizar.

Teniendo en cuenta las ventajas y desventajas de cada geometría y debido a que **N-CD++** modela solamente grillas con geometría cuadrada se desarrollo una herramienta que utilizando funciones de mapeo de geometría hexagonal a cuadrada y de geometría triangular a cuadrada **[Wei97]** permita simular modelos donde es indispensable contar con otras geometrías de celdas.

3.2.1 Función de mapeo de grilla hexagonal a cuadrada

Se pueden encontrar algunas funciones de mapeo, que si bien son aplicables, a la hora de la visualización de los resultados no es fácil la interpretación de los mismos. Otro problema que también se puede encontrar es la definición de la vecindad en los limites de la grilla. La idea gráficamente hablando se puede apreciar en la Figura 4.

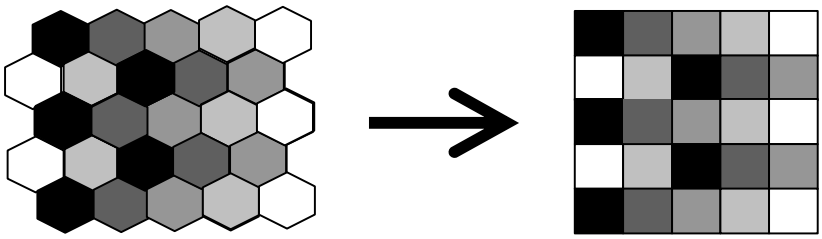


Figura 4 - Mapeo de geometría hexagonal a cuadrada

Como claramente se ve, la función de mapeo debería transformar las celdas hexagonales en las correspondientes cuadradas del mismo color.

La ventaja que se tendría con esta función de mapeo es que se seguiría utilizando las mismas condiciones de limites-vecindad que se usa en las grillas con geometría cuadrada.

La transformación se llevara a cabo dependiendo de la paridad o no de la fila en donde este la celda en cuestión. Se considerará que una celda se la ubica por su posición **(x,y)** dentro de la grilla, donde **x** es la fila e **y** es la columna (Figura 5).

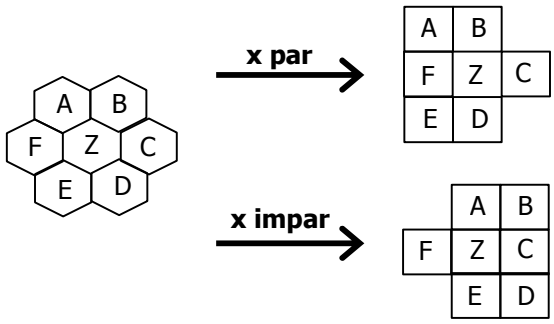


Figura 5 - Distribución de los vecinos cercanos en el mapeo

Como se puede ver las letras se corresponden de dos maneras diferentes, esto en realidad contradice la homogeneidad de las reglas en un autómata celular. Para esto se necesitará de funciones auxiliares que determinen la posición de la celda en cuestión. Se puede observar, analizando los colores de las celdas, en la Figura 4 que esta vecindad cubre todo el espacio celular.

3.2.2 Función de mapeo de grilla triangular a cuadrada

Este caso es similar al anterior. En la grilla triangular cada celda tiene una diferente orientación que sus vecinas. Para poder entender esto la idea gráfica se puede visualizar en la Figura 6.

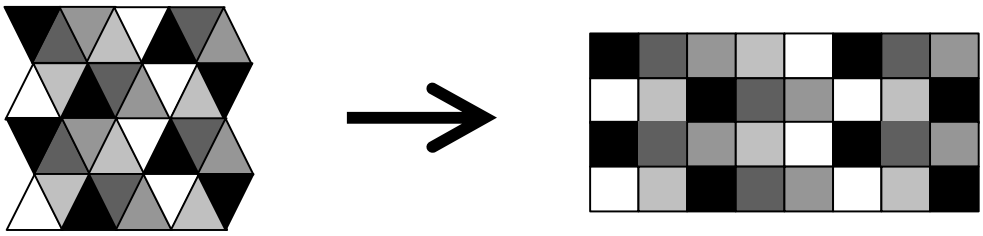


Figura 6 - Mapeo de geometría triangular a cuadrada

En esta función se mapea cada fila de triángulos con una fila de cuadrados, como se ve nuevamente no se tiene una vecindad uniforme, en este caso depende de la paridad de la suma de **x+y**, es decir de la paridad de la suma de la fila mas la columna (Figura 7):

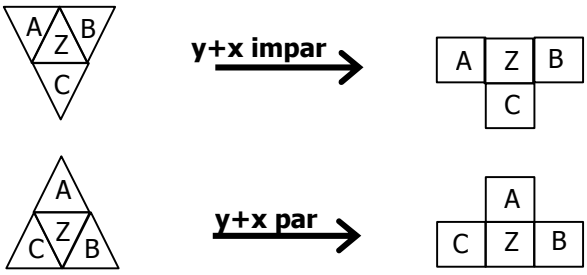


Figura 7 - Distribución de los vecinos cercanos en el mapeo

3.2.3 Ltrans: un traductor de grillas con geometría triangular o hexagonal a cuadrada

Como se menciono anteriormente **N-CD++** posee un lenguaje de especificación de reglas que definen el comportamiento de las celdas de geometría cuadrada, y esto hace que en la herramienta no se puedan simular una gran cantidad de modelos. Para poder revertir esta limitante se desarrollo una herramienta que permita mapear las reglas de especificación de comportamiento de celdas de una geometría hexagonal o triangular a geometría cuadrada. La misma se denomina **Ltrans** del ingles "Lattice Translator". Básicamente **Ltrans** toma una especificación de reglas basadas en grillas triangulares o hexagonales y lo traduce al lenguaje de especificación utilizado por el **N-CD++** utilizando las funciones de mapeo antes descriptas.

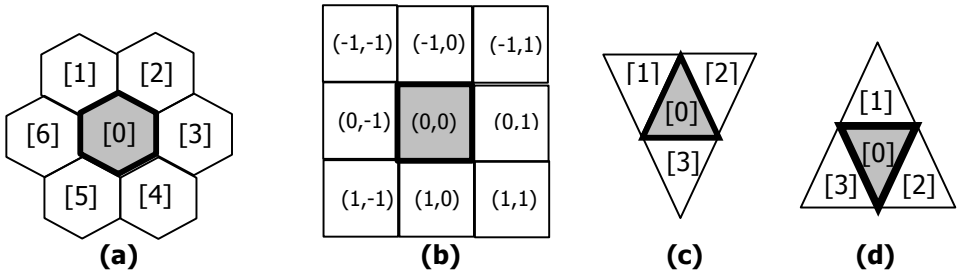


Figura 8 - Posición relativa de los vecinos en las distintas geometrías

El lenguaje de especificación para el comportamiento de las reglas en **Ltrans** difiere del leguaje utilizado por el **N-CD++** en la forma de referenciar a los vecinos más cercanos. En la Figura 8(b) podemos ver como **N-CD++** referencia cada celda de manera relativa a la celda en cuestión. En cambio en **Ltrans** Figura 8(a), 8(c) y 8(d) a los vecinos se le asigna un número fijo entre corchetes.

La función principal de **Ltrans** es la reescritura de las reglas para que **N-CD++** pueda llevar a cabo la simulación, esto se puede realizar debido al extenso conjunto de funciones que posee. Se utilizo la función que devuelve la i-esima posición dentro de la tupla que referencia a la celda donde se esta ejecutando la función de transición, y las dos funciones llamadas **odd** y **even** que al pasarle un número real como argumento devuelve un valor de verdad correspondiente a si ese número real es impar o par respectivamente. Luego por cada regla de comportamiento de una celda con geometría hexagonal o triangular le corresponden dos reglas en una geometría cuadrada dado que se debe tener en cuenta la paridad o no de las filas y columnas.

```
Rule: 1 10 {[3]=1 and statecount(9.99)=3}

Rule:   1   10   {(((0,1)=1)   and   (if((statecount(9.99)-(if((-1,1)=9.99,1,0))-
(if((1,1)=9.99,1,0)))<0,0,(statecount(9.99)-(if((-1,1)=9.99,1,0))-
(if((1,1)=9.99,1,0))))=3)) and even(cellpos(0))}

Rule: 1 10 {(((0,1)=1) and (if((statecount(9.99)-(if((-1,-1)=9.99,1,0))-(if((1,-
1)=9.99,1,0)))<0,0,(statecount(9.99)-(if((-1,-1)=9.99,1,0))-(if((1,-
1)=9.99,1,0))))=3)) and odd(cellpos(0))}
```

Figura 9 – Ejemplo de una regla en una geometría hexagonal con su correspondiente traducción

```
Rule: 1 10 {[3]=1 and statecount(9.99)=3}

Rule: 1 10 {(((0,-1)=1) and (if((statecount(9.99)-(if((-1,-1)=9.99,1,0))-(if((-
1,0)=9.99,1,0))-(if((-1,1)=9.99,1,0))-(if((1,-1)=9.99,1,0))-
(if((1,1)=9.99,1,0)))<0,0,(statecount(9.99)-(if((-1,-1)=9.99,1,0))-(if((-
1,0)=9.99,1,0))-(if((-1,1)=9.99,1,0))-(if((1,-1)=9.99,1,0))-
(if((1,1)=9.99,1,0))))=3)) and even(cellpos(0)+cellpos(1))}

rule: 1 10 {(((1,0)=1) and (if((statecount(9.99)-(if((-1,-1)=9.99,1,0))-(if((-
1,1)=9.99,1,0))-(if((1,-1)=9.99,1,0))-(if((1,0)=9.99,1,0))-
(if((1,1)=9.99,1,0)))<0,0,(statecount(9.99)-(if((-1,-1)=9.99,1,0))-(if((-
1,1)=9.99,1,0))-(if((1,-1)=9.99,1,0))-(if((1,0)=9.99,1,0))-
(if((1,1)=9.99,1,0))))=3)) and odd(cellpos(0) + cellpos(1))}
```

Figura 10 – Ejemplo de una regla en una geometría triangular con su correspondiente traducción

En la Figura 9 se puede apreciar la definición de una regla que será evaluada si: el vecino denominado [3] (ver Figura 8) tiene el valor 1, existen 3 celdas entre los vecinos y a su vez la misma celda (donde se esta evaluando la regla) tiene el valor 9.99. Luego, la celda tomara el valor 1 una vez transcurridos los 10 milisegundos de la demora. A continuación se pueden ver las dos reglas que son el resultado de la traducción. Para poder realizar la traducción debemos utilizar los valores correspondientes a la Figura 5 y 8(a), donde se detalla la correspondencia celda a celda de una grilla a la otra.

La función **statecount(n)** en una geometría cuadrada devuelve la cantidad de celdas entre las ocho celdas vecinas cercanas con el valor n. Esto hace que se deba tener en cuenta, por ejemplo en una geometría hexagonal, solo los seis vecinos cercanos correspondientes al mapeo (Figura 5 y 7). Por esto cuando traducimos esta función de una grilla a otra debemos tener en cuenta de que no contabilice aquellos vecinos que no correspondan. En la Figura 9 podemos ver como a la función **statecount(9.99)** se le descuenta los valores de las dos celdas ((-1,1) y (1,1)), dado que en las filas pares estas celdas no pertenecen a la vecindad cercana en una grilla hexagonal. Esa regla será evaluada si además la celda esta

en una fila par. Por ultimo la siguiente regla será evaluada si se cumple lo mismo que en la regla anterior (descontando ahora los dos vecinos que no se tienen en cuenta para las filas impares) pero con la diferencia que la celda debe estar en una fila impar. Esto ultimo se puede chequear con la función **cellpos(n)** que devuelve el n-esimo valor dentro de la tupla de referencia de una celda.

En la Figura 10 tenemos los resultados de aplicar la traducción a la misma regla pero ahora sobre la base de una geometría triangular. La diferencia principal radica no solo en la menor cantidad de vecinos cercanos posee esta geometría, sino que se debe controlar si la suma de los valores correspondientes a la fila y columna son par o impar.

En conclusión con dos simples funciones de mapeo se pueden desarrollar modelos celulares con geometría hexagonal o triangular, simularlos y observar los resultados con un pequeño error de visualización (debido a que se utilizan diferentes celdas vecinas dependiendo la paridad de filas y columnas).

3.3 *GrafLOG* : una forma de ver los resultados de las simulaciones de modelos celulares bidimensionales en modo gráfico

N-CD++ devuelve como resultado de la simulación un archivo de LOG donde se detalla la actividad del modelo o modelos simulados paso a paso. A su vez se puede utilizar una herramienta denominada **Drawlog** que permite representar gráficamente la actividad del simulador en cada instante del tiempo para los modelos celulares.

Drawlog es capaz de generar tres tipos de salidas, dependiendo de la dimensión del modelo celular a representar. Para los modelos de dos dimensiones se generará una matriz bidimensional con los valores de estado en cada instante del tiempo simulado, es decir se tendrá una matriz de números reales.

Para ayudar en la interpretación de los resultados en determinados modelos es necesario poder representar gráficamente con colores esos valores numéricos, dado que, de esta forma es mucho más fácil de entender el modelo. **GrafLOG** toma el archivo devuelto por el **Drawlog** y lo convierte a modo gráfico utilizando una tabla de asignación de colores a intervalos de números reales. En la Figura 11(a) y 11(b) se puede ver los resultados arrojados por las dos herramientas y las diferencias de visualización.

```
Line : 5414 - Time: 00:00:08:595
     0       1       2       3       4       5       6       7       8       9
 +-------------------------------------------------------------------------------+
0| 25.096  35.726  44.794  37.260  27.761  17.335  14.243  12.358  12.344  14.448|
1| 35.308  54.199  71.102  56.346  38.993  20.387  16.334  13.837  13.769  16.448|
2| 44.159  70.880  95.848  74.724  50.652  24.760  20.044  16.580  15.658  17.937|
3| 36.684  56.106  74.666  69.569  61.622  51.251  39.088  27.277  17.572  18.512|
4| 27.432  38.905  50.695  61.691  70.318  76.093  57.062  37.246  18.748  17.958|
5| 17.335  20.572  25.043  51.510  76.242  98.064  72.483  45.270  18.605  16.496|
6| 14.573  16.901  20.730  39.682  57.442  72.632  55.281  36.077  16.988  14.523|
7| 12.934  14.741  17.646  28.192  37.840  45.529  36.147  25.350  14.668  12.837|
8| 12.978  14.809  16.898  18.638  19.433  18.888  17.031  14.610  12.687  12.120|
9| 14.866  17.280  18.977  19.417  18.525  16.682  14.434  12.597  11.898  12.781|
 +-------------------------------------------------------------------------------+
```

Figura 11(a) – Resultado obtenido luego de correr el Drawlog

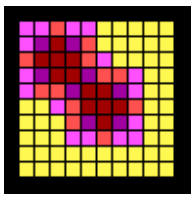


Figura 11(b) – El mismo resultado en modo gráfico utilizando el GrafLOG

El **GrafLOG** solo soporta 16 colores diferentes y cada intervalo puede ser abierto o cerrado, es decir que contenga o no a los números que definen el mismo. Además se agregaron dos constantes **–INF** e **+INF** para denotar a menos infinito y mas infinito respectivamente. En el apéndice se encuentra el manual del usuario.

4. Aplicaciones simples

En esta sección se describe como implementar diferentes modelos Cell-DEVS utilizando las herramientas antes descriptas.

4.1 Medio excitable

El primer caso de estudio es un modelo de medios excitables. Este fenómeno aparece en varios sistemas reales tales como, campos magnéticos, incendios forestales o por ejemplo las células del músculo del corazón (donde una onda de excitación viaja a través del corazón en cada latido). Si se considera por ejemplo las células que componen el músculo cardiaco, se tendrán tres estados en cada celda: Rojo (valor=2) significa excitada o en expansión, Gris (valor = 1) significa recuperándose y Blanco (valor=0) significa en descanso. Otros significados en otros modelos podrían ser, Blanco sin fuego, rojo significa que la celda se esta quemando y Gris significa que la celda se quemo.

El modelo puede ser especificado mediante el formalismo Cell-DEVS, debido a que se puede construir un modelo celular, donde cada celda del espacio puede ser formalmente definida por:

$$C_{ij} = < X, Y, I, demora, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

donde

$I = <9,0,P^x,P^y>$

$P^x = \{ (X_1,real), (X_2,real), (X_3,real), (X_4,real), (X_5,real), (X_6,real), (X_7,real), (X_8,real), (X_9,real)\}$

$P^y = \{ (Y_1,real), (Y_2,real), (Y_3,real), (Y_4,real), (Y_5,real), (Y_6,real), (Y_7,real), (Y_8,real), (Y_9,real)\}$

demora = transporte

S:

Variables descriptivas

$$s = \begin{cases} 2 \text{ si la celda esta excitada} \\ 1 \text{ si la celda se esta recuperándose} \\ 0 \text{ de otro modo la celda esta en reposo} \end{cases}$$

Parámetros:

extmp = demora relacionada con la posibilidad que una celda pueda cambiar de estado será en todo los caso de 100 milisegundos

$N = \{(-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1) (1,0), (1,1)\}$

$d = extmp$

$\tau : S \times N \rightarrow S$

$[((0,0) = 0 \text{ and statecount}(2) = 0) \text{ or } ((0,0)=1)] \rightarrow 0$

$[(0,0) = 2] \rightarrow 1$

$[(0,0) = 0 \text{ and statecount}(2) > 0] \rightarrow 2$

$[(0,0) = 0 \text{ or } (0,0) = 1 \text{ or } (0,0) = 2] \rightarrow \text{value}((0,0))$

Luego esta especificación puede escribirse en N-CD++ como se ve en la Figura 12.

```
01      [top]
02      components : ExMedia
03
04      [ExMedia]
05      type : cell
06      dim : (9,9)
07      delay : transport
08      defaultDelayTime  : 100
09      border : wrapped
10      neighbors : ExMedia(-1,-1) ExMedia(-1,0) ExMedia(-1,1)
11      neighbors : ExMedia(0,-1)  ExMedia(0,0)  ExMedia(0,1)
12      neighbors : ExMedia(1,-1)  ExMedia(1,0)  ExMedia(1,1)
13      initialvalue : 0
14      initialCellsValue : ExMedia.val
15      localtransition : calculus
16
17      [calculus]
18      rule : 0 100 { (0,0) = 0 and statecount(2) = 0 }
19      rule : 2 100 { (0,0) = 0 and statecount(2) > 0 }
20      rule : 1 100 { (0,0) = 2 }
21      rule : 0 100 { (0,0) = 1 }
22      rule : { (0,0) } 100 { t }
```

Figura 12 – Implementación de medios excitables

En la línea 1 y 2 se define el único modelo Cell-DEVS denominado ExMedia. En la línea 4 se comienza a describir el modelo. La línea 5 le indica al **N-CD++** que es un modelo celular, en la línea 6 se define la dimensión del espacio celular como así también las dimensiones del mismo. En este caso tendremos un espacio celular bidimensional de 9x9 celdas. La línea 7 define el tipo de demora que tendrá cada celda (ver Sección 2) en este caso particular la demora será de **transporte**. Luego en la línea 8 se define el tiempo de demora por defecto, en este caso se definió una demora de 100 milisegundos. La línea 9 define el tipo de borde dentro del espacio celular, en este caso será wrapped o toroidal. De la línea 10 a la 12 se define las posiciones relativas de los vecinos, en este caso el vecindario serán todos los vecinos cercanos incluyendo la celda que se esta analizando. A este tipo de vecindad se la denomina vecindad de Moore. La línea 13 define el valor por defecto que tendrá cada celda al comenzar la simulación, en este caso el valor inicial es 0 (en descanso). La línea 14 indica el archivo de valores iniciales, este archivo se utiliza para indicar valores especiales en celdas determinadas al inicio de la simulación. La línea 15 define el nombre de la sección que contendrá las reglas de comportamiento del modelo celular.

Una vez definido los parámetros que definen el contexto inicial de la simulación se deben escribir las reglas de comportamiento. En la línea 18 se encuentra la primer regla del modelo, será evaluada si cumple con las siguientes condiciones: el valor de la celda debe ser 0, es decir, debe estar en descanso y además todos los vecinos deben estar en algún estado distinto a 2 (excitada). Si estas condiciones se cumplen la celda toma el nuevo valor de 0 después de 100 milisegundos. En la línea 19 la regla se evaluará si cumple con: tener el valor 0 y tener por lo menos un vecino en estado 2 (excitada). Esto hace que el

nuevo valor sea 2 (excitada). Para entender esta regla simplemente se puede pensar en que una celda que esta en reposo se excitara si alguno de sus vecinos esta excitado, de alguna manera el vecino le esta transmitiendo el estado. La regla 20 será evaluada si la celda que se esta analizando está en estado 2 (excitada). Esto esta indicando que una celda en estado 2, permanecerá 100 milisegundos en ese estado y pasara al estado 1 (en recuperación). La línea 21 será evaluada si la celda tiene el valor 1(en recuperación), tomando como nuevo valor el 0 (n descanso). Por ultimo la línea 22 nos indica que si ninguna regla anterior no fue evaluada la celda tomara como nuevo valor el valor que posee en el momento de la evaluación de la regla.

En conclusión una celda pasa de estar en descanso a estar excitada si alguno de sus vecinos esta excitado, luego de 100 milisegundos pasa a estar en recuperación, para finalmente después de 100 milisegundos pasar a estar en descanso. En la Figura 13(a) se pueden ver el resultado en modo gráfico.

La Figura 13(b) es el resultado del mismo modelo pero cambiando la cantidad de vecinos. En este caso se definieron solo los vecinos que se encuentran al norte, sur, este y oeste (vecindad de Von Neuman). Por ultimo en la Figura 13(c) se ven los resultados obtenidos del modelo de medios excitables especificado sobre una geometría hexagonal y traducido con **Ltrans** a una grilla de geometría cuadrada.

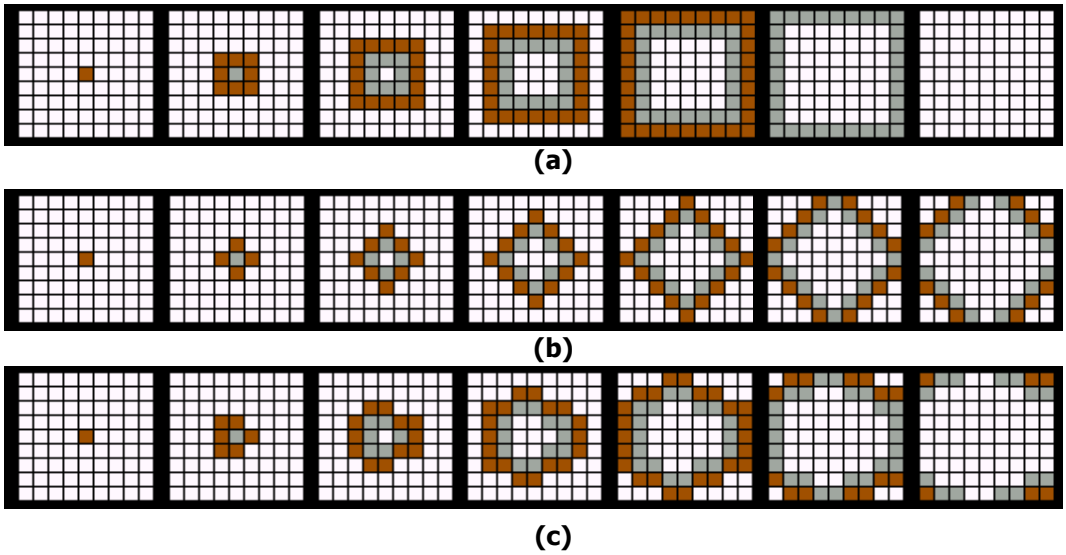


Figura 13 – Resultados de ExMedia con diferentes vecindades: (a) Moore, (b) Von Neuman y (c) Grilla hexagonal.

4.2 Tensión superficial

El segundo caso de estudio es un modelo de tensión superficial. En este fenómeno las partículas se van agrupando en los lugares donde hay mayor tensión. El modelo puede ser representado como un sistema de "mayoría alineada". Existen dos estados posibles, 1 representa la presencia y 0 representa la ausencia de partículas.

De manera similar al modelo anterior, se puede especificar el modelo como un modelo celular donde cada celda del espacio puede ser formalmente definida por:

$$C_{ij} = < X, Y, I, demora, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

donde

$I = <9,0,P^x,P^y>$

$P^x = \{ (X_1,real), (X_2,real), (X_3,real), (X_4,real), (X_5,real), (X_6,real), (X_7,real), (X_8,real), (X_9,real)\}$

P^y = { (Y_1,real), (Y_2,real), (Y_3,real), (Y_4,real), (Y_5,real), (Y_6,real), (Y_7,real), (Y_8,real), (Y_9,real)}

demora = transporte

S:

Variables descriptivas

s = { 1 indica la presencia de una partícula en la celda recuperándose
{ 0 de otro modo

Parámetros:

tenstmp = demora relacionada con el cambio de tensión en la superficie en todo los caso esta demora es de 100 milisegundos

N = {(-1,-1) , (-1,0), (-1,1), (0,-1), (0,0) , (0,1), (1,-1) (1,0), (1,1)}

d = *tenstmp*

τ : S x N $\rightarrow$ S
[statecount(0) > 5] $\rightarrow$ 0
[statecount(0) <= 5] $\rightarrow$ 1

```
01      [top]
02      components : tensionsup
03
04      [tensionsup]
05      type : cell
06      dim : (40,40)
07      delay : transport
08      defaultDelayTime  : 100
09      border : wrapped
10      neighbors : tensionsup(-1,-1) tensionsup(-1,0) tensionsup(-1,1)
11      neighbors : tensionsup(0,-1)  tensionsup(0,0)  tensionsup(0,1)
12      neighbors : tensionsup(1,-1)  tensionsup(1,0)  tensionsup(1,1)
13      initialvalue : 0
14      initialCellsValue : tension.val
15      localtransition : calculus
16
17      [calculus]
18      rule : 0 100 { statecount(0) >= 5 }
19      rule : 1 100 { t }
```

Figura 14 – Implementación del modelo de tensión superficial

En la Figura 14 se puede ver la especificación completa del modelo en la herramienta. El la línea 1 y 2 se definen el único componente que conforma el modelo, denominado tensionsup. A partir de la línea 4 se comienza con la definición del modelo. Nuevamente en la línea 5 se define el tipo de componente, en este caso es un modelo Cell-DEVS. En la línea 6 se define un espacio bidimensional de 40x40 celdas. En la línea 7 se define el tipo de demora, se asigno una demora de transporte. En la línea 8 se define el tipo de demora para el cambio de estado de cada celda, por defecto será de 100 milisegundos. En la línea 9 se selecciono el tipo de borde wrapped o toroidal. En las líneas 10, 11 y 12 se define la vecindad, nuevamente se selecciona la vecindad de Moore, que corresponde a los ocho vecinos cercanos. En la línea 13 se define el valor de cada celda que se tomara al iniciar la simulación por defecto. En la línea 14 se define el nombre del archivo que contiene valores específicos para determinadas celdas. En este caso se opto por una distribución de particular aleatorias a lo largo de la superficie. El la línea 15 se asigna el nombre de la

sección que contendrá las reglas que rigen los comportamientos de cada celda del modelo. En la línea 18 se encuentra la primer regla, asigna el valor 0 a la celda que se esta evaluando si posee cinco o mas celdas vecinas sin partículas, es decir, una celda pasa a estar en 0 si tiene tres o menos vecinos que contengan una partícula. Por ultimo, para cualquier caso que no cumpla con la primer regla se le asigna el valor 1 (presencia de una partícula). En conclusión este modelo agrupa las partículas en aquellos lugares donde exista una mayoría de partículas.

En la Figura 15 podemos apreciar el comportamiento del modelo a lo largo del avance del tiempo en la simulación. La Figura 15(a) presenta una distribución aleatoria de particular, luego de transcurrido el tiempo podemos ver en la Figura 15(f) como las partículas se concentraron en los puntos donde de un principio existían mas partículas.

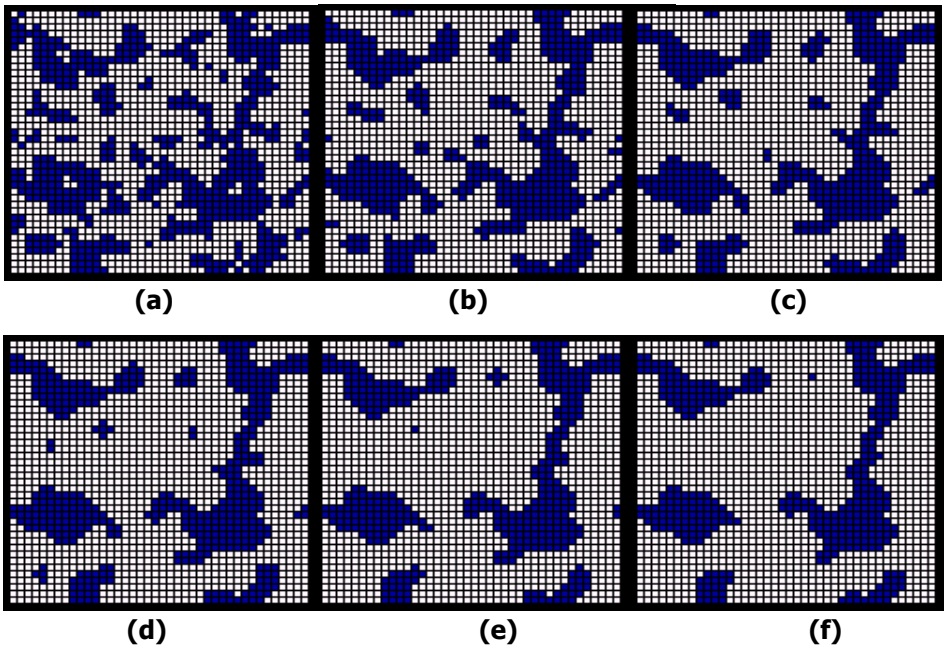


Figura 15 – Resultados de la ejecución del modelo de tensión superficial

4.3 Difusión de calor

El siguiente caso de estudio es sobre la difusión de calor a lo largo de una superficie. Este es un multimodelo donde se incluye tres componentes: la superficie, un generador de calor y un generador de frío. En este modelo se puede apreciar las diferentes temperaturas en la superficie representadas por diferentes colores que van desde el rojo que representa altas temperaturas al azul que representa bajas temperaturas. En cada etapa de la simulación la temperatura de cada celda es calculada como el promedio de los valores de los ocho vecinos que rodean a la misma.

Debido a que es un modelo Cell-DEVS bidimensional acoplado con otros modelos DEVS, el mismo se puede definir como:

$$CC = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, \eta, N, C, B, Z, select >$$

donde:

Xlist = {(5,5), (2,2), (8,8), (2,8)} corresponde a las celdas donde existe acoplamiento interno

Ylist = {∅} debido a que no tiene acoplamiento externo

$I = < P^x, P^y >$ donde:

P^x = { <X(5,5), Real>, <X(2,2), Real> , <X(8,8), Real>, <X(2,8), Real>}
P^y = {∅}

n = 2

$\{t_1=10, t_2=10\}$

$\eta = 10$

$X \subseteq \Re$

$Y \subseteq \Re$

N = {(-1,-1) , (-1,0), (-1,1), (0,-1), (0,0) , (0,1), (1,-1) (1,0), (1,1)}

$C = \{C_{k1, k2, ..., kn} / k_i \in [0,t_i] \ \forall \ i \in N, i \in [1,n]\}$, es el conjunto de celdas atómicas que conforman al modelo acoplado, donde:

C_{ij} = < *X*, *Y*, I, demora, S, θ, N, d, δ_{int}, δ_{ext}, τ, λ, D >

donde

I = <9,0,P^x,P^y>
P^x = { (X_1,real), (X_2,real), (X_3,real), (X_4,real), (X_5,real), (X_6,real), (X_7,real), (X_8,real), (X_9,real)}
P^y = { (Y_1,real), (Y_2,real), (Y_3,real), (Y_4,real), (Y_5,real), (Y_6,real), (Y_7,real), (Y_8,real), (Y_9,real)}

demora = transporte

S:

Variables descriptivas

$s \in \Re$

Parámetros:

calortmp = demora relacionada con la difusión del calor a los vecinos. La misma es de 100 milisegundos

N = {(-1,-1) , (-1,0), (-1,1), (0,-1), (0,0) , (0,1), (1,-1) (1,0), (1,1)}

d = *calortmp*

$\tau : S \times N \rightarrow S$
[verdadero] → ((-1,-1)+(-1,0)+(-1,1)+(0,-1)+(0,0)+(0,1)+(1,-1)+(1,0)+(1,1))/9)

B = {∅} dado que el espacio de celdas es toroidal (Wrapped)

Luego esta especificación puede escribirse en N-CD++ como se ve en la Figura 16.

```
01      [top]
02      components : superficie generadorCalor@Generator generadorFrio@Generator
03      link : out@generadorCalor inputCalor@superficie
```

```
04      link : out@generadorFrio  inputFrio@superficie
05
06      [superficie]
07      type : cell
08      dim : (10,10)
09      delay : transport
10      defaultDelayTime  : 100
11      border : wrapped
12      neighbors : superficie(-1,-1) superficie(-1,0) superficie(-1,1)
13      neighbors : superficie(0,-1)  superficie(0,0)  superficie(0,1)
14      neighbors : superficie(1,-1)  superficie(1,0)  superficie(1,1)
15      initialvalue : 10
16      in : inputCalor inputFrio
17      link : inputCalor in@superficie(5,5)
18      link : inputCalor in@superficie(2,2)
19      link : inputFrio  in@superficie(8,8)
20      link : inputFrio  in@superficie(2,8)
21      localtransition : calor-rule
22      portInTransition : in@superficie(5,5)  setCalor
23      portInTransition : in@superficie(2,2)  setCalor
24      portInTransition : in@superficie(8,8)  setFrio
25      portInTransition : in@superficie(2,8)  setFrio
26
27
28      [calor-rule]
29      rule : { ((0,0) + (-1,-1) + (-1,0) + (-1,1) + (0,-1) + (0,1) + (1,-1) +
    (1,0) + (1,1)) / 9 } 1000 { t }
30
31      [setCalor]
32      rule : { uniform(524,640) } 1000 { t }
33
34      [setFrio]
35      rule : { uniform(-10,15) } 1000 { t }
36
37      [generadorCalor]
38      distribution : exponential
39      mean : 3
40      initial : 1
41      increment : 0
42
43      [generadorFrio]
44      distribution : exponential
45      mean : 50
46      initial : 1
47      increment : 0
```

Figura 16 – Implementación del modelo de difusión de calor

Como se menciona en el párrafo anterior, este es un multimodelo o un modelo acoplado. En la línea 1 y 2 se definen los componentes o modelos atómicos que conforman el mismo. Se tiene un componente que es la superficie, y otros dos componentes que son clases derivadas de un componente denominado generador. El primero es un componente Cell-DEVS, mientras que los otros dos son componentes DEVS. La línea 3 y 4 definen la interconexión entre componentes, la línea 3 especifica que el puerto de salida del generador de calor estará conectado al puerto denominado **inputCalor** ubicado la superficie. Lo mismo sucede en la línea 4, donde el generador de frío esta conectado a un puerto denominado **inputFrio** ubicado en la superficie.

En la línea 6 se comienza por la definición del modelo Cell-DEVS que representa a la superficie. En la línea 7 se define como un modelo celular. En la línea 8 se define un espacio celular bidimensional de 10x10 celdas. En la línea 9 se define como tipo de demora a la demora por transporte. En la línea 10 se define el tiempo de la demora por defecto igual a 100 milisegundos. Luego en la línea 11 se define el tipo de bordes que en este caso será de tipo toroidal o wrapped. De la línea 12 a 14 se define como vecindad a los ocho vecinos cercanos (vecindad de Moore). En la línea 15 se define como valor inicial para cada celda

de la grilla a 10ºC, este valor representara a la temperatura ambiente al comenzar la simulación. La línea 16 define la cantidad y el nombre de los puertos de entrada que tendrá ese modelo, en este caso se tendrá dos puestos de entrada: **inputCalor** e **inputFrio.** De la línea 17 a 20 se define como se conectaran internamente esos puertos de entrada, es decir, a que celdas estarán conectados. Como se ve en la línea 17 y 18 se conecta el puerto **inputCalor** a las celdas que están ubicadas en la posición (5,5) y (2,2). Por otro lado en las líneas 19 y 20 conectamos el puerto **inputFrio** a las celdas (8,8) y (2,8).

La línea 21 define la sección donde se encontraran las reglas que rige el comportamiento de cada celda. Luego de la línea 22 a 25 se definen las secciones donde se especifica el comportamiento de los puertos de entrada, es decir, que valores entraran por ese puerto. En la línea 28 se define la única regla que rige el comportamiento de cada celda. Como se puede ver la regla siempre es evaluada y el resultado es el promedio de los valores de los vecinos incluyendo a la misma celda. En al línea 31 se encuentra la sección que define el comportamiento para los valores que arriban por el puerto de entrada **inputCalor**. Se utilizo una distribución uniforme que asigna valores cada 1 segundo entre 524ºC y 640ºC. Por otro lado en la línea 34 y 35 se define los valores que arribaran por el puerto **inputFrio** que están comprendidos entre –10ºC y 15ºC.

En al línea 37 de define la segunda componente del modelo, en este caso es un modelo DEVS denominado **Generator** cuya función es generar un evento cada un intervalo predeterminado. Este intervalo es generado por una función probabilística. En el modelo se utiliza la función exponencial con media 3 en el caso del **generadorCalor** y media 5 en el caso del **generadorFrio**.

Por ultimo en la Figura 17 se puede apreciar el resultado de la ejecución del modelo. En los estado iniciales se ve como las celdas que están conectadas a los puertos cambian su temperatura y van disipando tanto el calor como el frío.

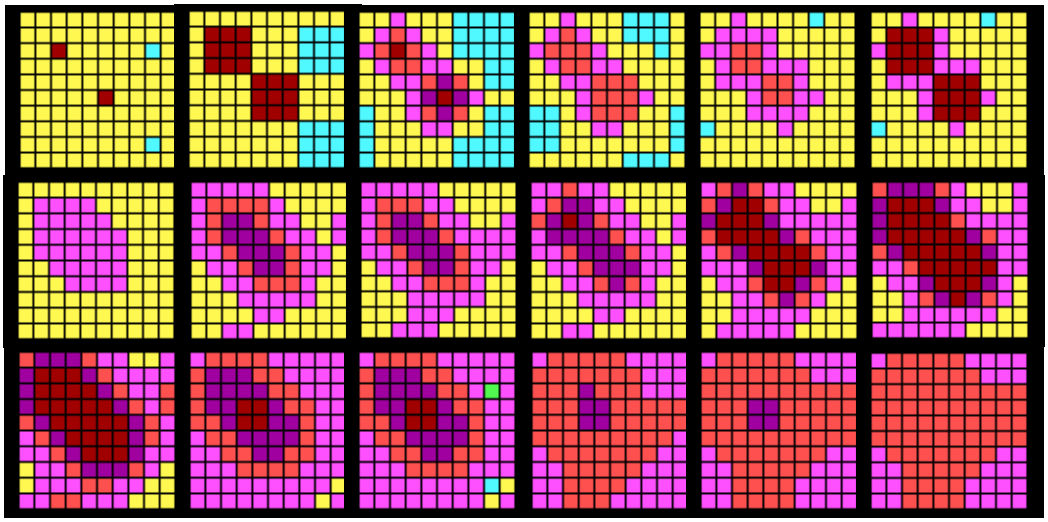


Figura 17 – Resultados de la ejecución del modelo de difusión de Calor

4.4 Colisión de partículas utilizando la Vecindad de Margolus.

Como se menciono anteriormente existen distintos tipos de vecindad en los autómatas celulares. En este caso se estudia un tipo de vecindad particular denominada vecindad de Margolus. Esta vecindad es utilizada en modelos físicos, y especialmente cuando se trata de modelos que sean reversibles, es decir, que se pueda volver hacia atrás en el tiempo.

En primer lugar se comienza describiendo un estilo de autómata celular bidimensional denominado **autómata celular particionado**. Estos estilo de autómatas celulares consisten en:

1. La grilla de celdas es particionada en una colección finita, disjunta y uniforme de bloques.

2. La regla para los bloques se define por el valor que tienen las celdas que componen dicho bloque y se actualiza de la misma manera (ver Figura 18). Dado que los bloques no se solapan no se tendrá intercambio de información entre bloques adyacentes.

3. La partición de la grilla es cambiada en cada paso con respecto a la anterior, dado que si se utiliza la misma partición siempre estaríamos en presencia de un autómata celular subdividido en una colección de bloques independientes.

Para que los ejemplos resulten más fáciles de visualizar se particiona la grilla en bloques de 2x2 celdas y se alterna la partición de la grilla en aquellas celdas que están alineadas con la grilla par y con la grilla impar (ver Figura 19).

Se debe tener en cuenta que la vecindad de Margolus hace uso de solamente dos particiones (llamadas grilla par y grilla impar), esto se debe a que las diferentes particiones usadas paso a paso por las reglas deben ser finitas en numero y deben ser rehusadas de manera cíclica manteniendo de alguna manera la uniformidad del espacio y tiempo.

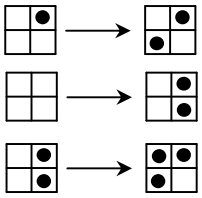


Figura 18 - Reglas de bloques de 2x2 celdas

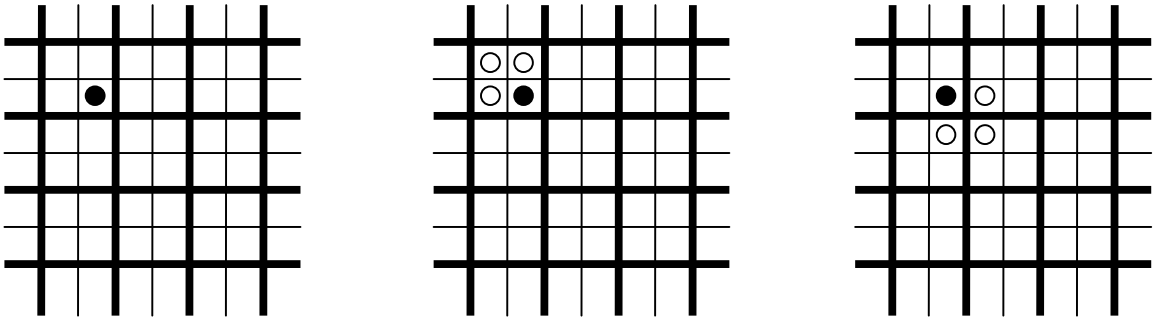


Figura 19 - Los bloques de 2x2 celdas correspondientes a la grilla par (líneas gruesas) y a la grilla impar (líneas finas). Dependiendo de cual grilla se este utilizando la celda marcada en (a) tendrá como vecinos a los que están alineadas con la grilla par (b) y a los que están alineadas con la grilla impar (c).

Como se menciono anteriormente la vecindad permanece constante a lo largo de toda la simulación, es decir, a lo largo de toda la simulación siempre los vecinos son los mismos. Pero en este tipo de vecindad se tienen dos maneras diferentes de analizar los valores de los vecinos y esto ocurre de manera alternada, aquellos que se alinean a la grilla par y aquellos que se alinean a la grilla impar.

Para poder aplicar este concepto sobre **N-CD++** se debe tener algún mecanismo que permita saber de manera alternada quienes son los vecinos en un instante determinado. Como la vecindad de Margolus se aplica en modelos celulares bidimensionales, y en **N-CD++** se pueden construir modelos celulares de n-dimensiones, se utiliza un modelo tridimensional, donde el plano 0 o plano superior es el

autómata celular y el plano 1 o plano inferior permite, según el valor de cada celda, saber si se debe tener en cuenta a los vecinos que están alineados a la grilla par o los que están alineados a la grilla impar.

Como se utilizan particiones de 2x2 celdas, se tiene que definir una vecindad en **N-CD++** que abarque a todas las posibilidades y esto se hace definiendo una vecindad de Moore (los ocho vecinos cercanos).

Por ultimo se cuenta con las funciones **odd**, **even** (que permiten determinar la paridad o no de determinado valor) y **cellpos** que permite proyectar cualquier coordenada dentro de la tupla de referencia a una celda.

Para poder ver los resultados se detalla a continuación un modelo de movimiento de partículas de gas que colisionan denominado HPP-Gas. El modelo define un mecanismo de movimiento uniforme de partículas. Estas partículas poseen masa y consecuentemente energía kinética, además son todas idénticas y se mueven todas a la misma velocidad dado que tienen la misma energía. Por ultimo se debe conservar la energía, por lo tanto se debe conservar las partículas. En la Figura 20 se detallan las reglas haciendo uso la vecindad de Margolus.

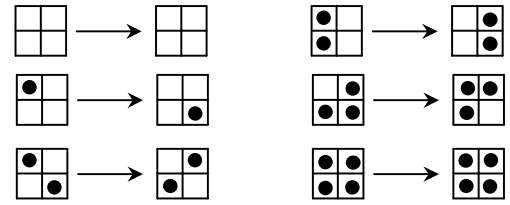


Figura 20 – Reglas del modelo HPP-Gas

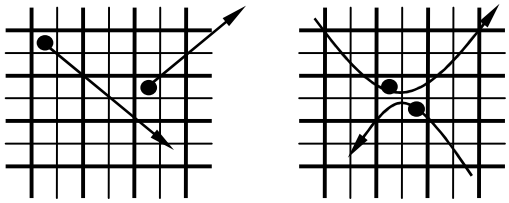


Figura 21 – HPP-Gas: (a) Movimiento uniforme de partículas aisladas. (b) Partículas que colisionan y cambian de dirección hacia la diagonal opuesta.

```
01      [top]
02      components : Collisions
03
04      [Collisions]
05      type : cell
06      dim : (40,40,2)
07      delay : transport
08      defaultDelayTime  : 100
09      border : wrapped
10      neighbors : collisions(-1,-1,0) collisions(-1,0,0) collisions(-1,1,0)
11      neighbors : collisions(0,-1,0)  collisions(0,0,0)  collisions(0,1,0)
collisions(0,0,1)
12      neighbors : collisions(1,-1,0)  collisions(1,0,0)  collisions(1,1,0)
13      initialvalue : 0
14      initialCellsValue : Colli.val
15      localtransition : calculus
16
17      [calculus]
```

```
18      rule : { (1,1,0) } 100 { cellpos(2)=0 and ((even(cellpos(0)) and
even(cellpos(1)) and (0,0,1)=0) or (odd(cellpos(0)) and odd(cellpos(1)) and
(0,0,1)=1)) }
19      rule : { (1,-1,0) } 100 { cellpos(2)=0 and ((even(cellpos(0)) and
odd(cellpos(1)) and (0,0,1)=0) or (odd(cellpos(0)) and even(cellpos(1)) and
(0,0,1)=1)) }
20      rule : { (-1,1,0) } 100 { cellpos(2)=0 and ((odd(cellpos(0)) and
even(cellpos(1)) and (0,0,1)=0) or (even(cellpos(0)) and odd(cellpos(1)) and
(0,0,1)=1)) }
21      rule : { (-1,-1,0) } 100 { cellpos(2)=0 and ((odd(cellpos(0)) and
odd(cellpos(1)) and (0,0,1)=0) or (even(cellpos(0)) and even(cellpos(1)) and
(0,0,1)=1)) }
22
23      rule : 1 100 { cellpos(2)=1 and (0,0,0)=0 }
24      rule : 0 100 { cellpos(2)=1 and (0,0,0)=1 }
25      rule : { (0,0,0) } 100 { t }
26
```

Figura 22 – Implementación del modelo HPP-Gas

En el modelo celular bidimensional utilizado para la especificación de la vecindad, cada celda del espacio celular puede ser formalmente definida por:

$$C_{ij} = < X, Y, I, \text{demora}, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

donde

$I = <9,0,P^x,P^y>$

$P^x = \{(X_1,\text{real}), (X_2,\text{real}), (X_3,\text{real}), (X_4,\text{real}), (X_5,\text{real}), (X_6,\text{real}), (X_7,\text{real}), (X_8,\text{real}), (X_9,\text{real})\}$

$P^y = \{ (Y_1,\text{real}), (Y_2,\text{real}), (Y_3,\text{real}), (Y_4,\text{real}), (Y_5,\text{real}), (Y_6,\text{real}), (Y_7,\text{real}), (Y_8,\text{real}), (Y_9,\text{real})\}$

demora = transporte

S:

Variables descriptivas

s = { 1 indica la presencia de una partícula en la celda
{ 0 de otro modo

Parámetros:

movtmp = demora relacionada con el recorrido de la particula de una celda a otra, esta demora es de 100 milisegundos

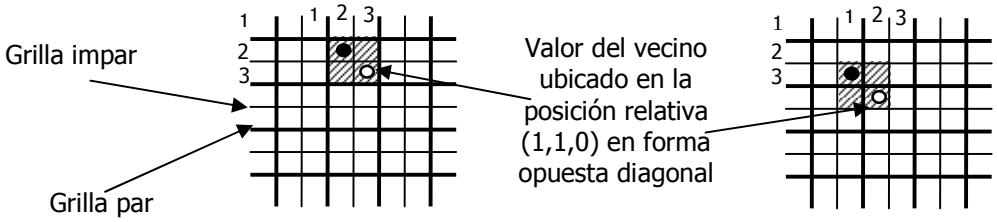
N = {(-1,-1) , (-1,0), (-1,1), (0,-1), (0,0) , (0,1), (1,-1) (1,0), (1,1)}

d = *movtmp*

La función de computo junto con la especificación completa de este modelo puede escribirse en N-CD++ como se ve en la Figura 22. En las líneas 1 y 2 se definen el único componente de este modelo. De la línea 4 a la línea 9 se define un modelo celular tridimensional de 40x40x2, es decir, dos planos de 40x40 celdas. El primero (Plano 0) refleja el movimiento y colisiones de partículas, mientras que el segundo plano (Plano 1) indica si se tiene que utilizar los vecinos alineados a la grilla par o a la grilla impar. La demora es una demora de transporte, y el tiempo por defecto de demora es de 100 milisegundos. Los bordes son toroidales o wrapped. En las líneas 10 a 12 se define la vecindad, en este caso se define para cada celda los ocho vecinos cercanos (vecindad de Moore) y además se le agrega un vecino que pertenece al plano inferior quien será, según su valor, quien nos indica que vecinos tomar. En la línea 13 y 14 se definen los valores por defecto de las celdas y además el nombre del archivo donde se puede asignar valores específicos a un conjunto de celdas en particular.

Si nuevamente se observa la Figura 20, se ve que cada celda tomara el valor de su vecino opuesto en dirección diagonal. Luego las líneas 18 a 21 definen para cada uno de las cuatro celdas que valor tomar y en que momento.

Analizando la línea 18 se puede ver que la regla será evaluada si se esta en presencia de dos situaciones diferentes: si la celda esta ubicada en una fila y columna par de la grilla y además se están analizando los vecinos que se alinean a la grilla par, o la celda esta ubicada en una fila y columna impar de



la grilla y además se están analizando los vecinos que se alinean a la grilla impar. En cualquiera de estas dos situaciones la celda actual toma el valor de su vecino ubicado en la posición relativa (1,1,0) como se puede ver en la Figura 23.

(a) **(b)**

Figura 23 – (a) Celda ubicada en una posición con fila y columna par alineada con la grilla par. (b) Celda ubicada en una posición con fila y columna impar alineada con la grilla impar

En las líneas 19, 20 y 21 el concepto es el mismo, donde la única diferencia es la posición de la celda que se esta evaluando en la partición.

Las líneas 23 y 24 sirven para que de manera alternada cada 100 milisegundos todas las celdas pertenecientes al plano 1 o plano inferior contengan el valor 0 y el 1, donde 0 servirá para tener en cuenta a las celdas alineadas a la grilla par y el 1 para las celdas alineadas a la grilla impar.

Por ultimo si ninguna regla anterior es evaluada, se evaluara la regla de la línea 25, que mantiene el valor en el siguiente paso.

4.5 Buscador de Calor

Este caso consiste de un modelo multicomponentes: un modelo celular de tres dimensiones, que representa una determinada área conteniendo diferentes temperaturas y dos generadores de calor. El modelo cuenta con un objeto que trata de encontrar la celda con mayor temperatura, moviéndose por el área representada por la grilla en 4 direcciones posibles. El objeto podría ser una cabeza de un misil con un sensor de calor cuyo objetivo es alcanzar a la zona de la grilla que tenga mas temperatura. El modelo contiene tres planos, en el plano 0 se modela el movimiento del misil, en el plano 1 se modela la temperatura y el plano 2 determina la etapa en la cual el modelo se encuentra.

Los posibles valores de estado para las celdas en el plano 0 son:

0: Indica que la celda se encuentra vacía.
1: Indica que la celda contiene al misil y se dirige en dirección norte.
2: La celda contiene al misil y se dirige en dirección este.
3: La celda contiene al misil y se dirige en dirección sur.
4: La celda contiene al misil y se dirige en dirección este.
10: Indica que la celda será ocupada por el misil en el siguiente paso en dirección norte.
20: La celda será ocupada por el misil en el siguiente paso en dirección este.
30: La celda será ocupada por el misil en el siguiente paso en dirección sur.
40: La celda será ocupada por el misil en el siguiente paso en dirección este.

Los posibles valores en el plano 1, serán valores de temperatura que variaran entre los 0ºC y los 95ºC.

Por otro lado los valores posibles en el plano 2 serán:

0: Indica que se debe calcular la nueva dirección del misil analizando la temperatura
1: Se debe calcular la nueva posición que ocupara el misil en el próximo paso
2: Se debe colocar a valor a cero a la posición donde estaba el misil antes de avanzar
3: Se debe tener solo valores de celda entre 0 y 4.
4..6: Se deben actualizar los valores de la temperatura en el plano 1

Debido a que es un modelo Cell-DEVS tridimensional dado que se tienen dos planos (y acoplado con otros dos modelos DEVS) se puede definir formalmente como:

$$CC = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, \eta, N, C, B, Z, select >$$

donde:

Xlist = {(6,2,1), (2,7,1) } corresponde a las celdas donde existe acoplamiento interno

Ylist = {∅} debido a que no tiene acoplamiento externo

$I = < P^x, P^y >$ donde:
P^x = { <X(6,2,1), Real>, <X(2,7,1), Real>}
P^y = {∅}

n = 3

$\{t_1=10, t_2=10, t_2=3 \}$

$\eta = 19$

$X \subseteq \Re$

$Y \subseteq \Re$

N = {(-1,-1,0) , (-1,0,0), (-1,1,0), (0,-1,0), (0,0,0) , (0,1,0), (1,-1,0) (1,0,0), (1,1,0), (-1,-1,1) , (-1,0,1), (-1,1,1), (0,-1,1), (0,0,1) , (0,1,1), (1,-1,1) (1,0,1), (1,1,1), (0,0,2)}

$C = \{C_{k1, k2, ..., kn} / k_i \in [0,t_i] \ \forall \ i \in N, i \in [1,n]\}$, es el conjunto de celdas atómicas que conforman al modelo acoplado, donde:

$$C_{ij} = < X, Y, I, demora, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

donde

$I = <19,0,P^x,P^y>$
P^x = { (X_1,real), (X_2,real), (X_3,real), (X_4,real), (X_5,real), (X_6,real), (X_7,real), (X_8,real), (X_9,real), (X_{10},real), (X_{11},real), (X_{12},real), (X_{13},real), (X_{14},real), (X_{15},real), (X_{16},real), (X_{17},real), (X_{18},real), (X_{19},real)}
P^y = { (Y_1,real), (Y_2,real), (Y_3,real), (Y_4,real), (Y_5,real), (Y_6,real), (Y_7,real), (Y_8,real), (Y_9,real), (Y_{10},real), (Y_{11},real), (Y_{12},real), (Y_{13},real), (Y_{14},real), (Y_{15},real), (Y_{16},real), (Y_{17},real), (Y_{18},real), (Y_{19},real)}

demora = transporte

S:

Variables descriptivas

$s \in \Re$

Parámetros:

calortmp = demora relacionada con la difusión del calor a los vecinos. La misma es de 100 milisegundos

N = {(-1,-1,0) , (-1,0,0), (-1,1,0), (0,-1,0), (0,0,0) , (0,1,0), (1,-1,0) (1,0,0), (1,1,0), (-1,-1,1) , (-1,0,1), (-1,1,1), (0,-1,1), (0,0,1) , (0,1,1), (1,-1,1) (1,0,1), (1,1,1), (0,0,2)}

d = *calortmp*

B = {∅} dado que el espacio de celdas es toroidal (Wrapped)

Esta especificación completa puede escribirse en N-CD++ como se ve en la Figura 24, la función de computo local esta representada en la sección **calculus**.

```
01      [top]
02      components : Seeker
03      in : inPort1 inPort2
04      link : inPort1 inputCalor1@Seeker
05      link : inPort2 inputCalor2@Seeker
06
07      [Seeker]
08      type : cell
09      dim : (10,10,3)
10      delay : transport
11      defaultDelayTime  : 100
12      border : wrapped
13      neighbors : Seeker(-1,-1,0) Seeker(-1,0,0) Seeker(-1,1,0)
14      neighbors : Seeker(0,-1,0)  Seeker(0,0,0)  Seeker(0,1,0)
15      neighbors : Seeker(1,-1,0)  Seeker(1,0,0) Seeker(1,1,0)
16
17      neighbors : Seeker(-1,-1,1) Seeker(-1,0,1) Seeker(-1,1,1)
18      neighbors : Seeker(0,-1,1)  Seeker(0,0,1)  Seeker(0,1,1)
19      neighbors : Seeker(1,-1,1)  Seeker(1,0,1) Seeker(1,1,1)
20
21      neighbors :                 Seeker(0,0,2)
22      initialvalue : 0
23      in : inputCalor1 inputCalor2
24      link : inputCalor1 in@Seeker(6,2,1)
25      link : inputCalor2 in@Seeker(2,7,1)
26
27      initialCellsValue : Seeker.val
28      localtransition : calculus
29      portInTransition : in@Seeker(6,2,1)  setCalor1
30      portInTransition : in@Seeker(2,7,1)  setCalor2
31
32      [calculus]
33      rule : 2 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=1 and ((0,-1,1) -
(0,1,1)) < 0 }
34      rule : 4 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=1 and ((0,-1,1) -
(0,1,1)) > 0 }
35      rule : 3 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=2 and ((-1,0,1) -
(1,0,1)) < 0 }
36      rule : 1 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=2 and ((-1,0,1) -
(1,0,1)) > 0 }
37      rule : 4 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=3 and ((0,1,1) -
(0,-1,1)) < 0 }
```

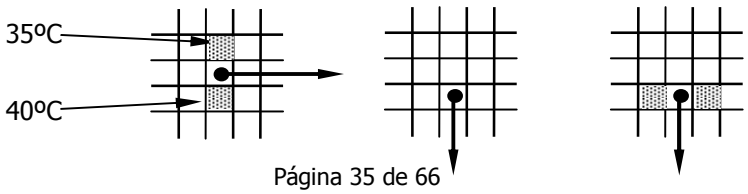
```
38      rule : 2 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=3 and ((0,1,1) -
(0,-1,1)) > 0 }
39      rule : 1 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=4 and ((1,0,1) - (-
1,0,1)) < 0 }
40      rule : 3 100 { cellpos(2)=0 and (0,0,2)=0 and (0,0,0)=4 and ((1,0,1) - (-
1,0,1)) > 0 }
41
42      rule : 10 100 { cellpos(2)=0 and (0,0,2)=1 and (1,0,0)=1 }
43      rule : 20 100 { cellpos(2)=0 and (0,0,2)=1 and (0,-1,0)=2 }
44      rule : 30 100 { cellpos(2)=0 and (0,0,2)=1 and (-1,0,0)=3 }
45      rule : 40 100 { cellpos(2)=0 and (0,0,2)=1 and (0,1,0)=4 }
46
47      rule : 0 100 { cellpos(2)=0 and (0,0,2)=2 and (0,0,0)<10 }
48
49      rule : {(0,0,0)/10} 100 { cellpos(2)=0 and (0,0,2)=3 and (0,0,0)>=10 }
50
51      rule : { ((-1,-1,0)+(-1,0,0)+(-1,1,0)+(0,-1,0)+(0,0,0)+(0,1,0)+(1,-
1,0)+(1,0,0)+(1,1,0))/9 } 100 {cellpos(2)=1 and (0,0,1)>=4}
52
53      rule : {(0,0,0)+1} 100 { cellpos(2)=2 and (0,0,0) < 6 }
54      rule : 0 100 { cellpos(2)=2 and (0,0,0)>=6 }
55
56      rule : { (0,0,0) } 100 { t }
57
58      [setCalor1]
59      rule : { uniform(30,65) } 0 { t }
60      [setCalor2]
61      rule : { uniform(65,95) } 0 { t }
```

Figura 24 – Implementación del modelo del buscador de calor

En las líneas 1 a 5 se definen los componentes del modelo y como se interconectan. El modelo celular se llama Seeker y luego se tienen dos puertos conectados al modelo celular donde se genera calor para atraer al buscador.

De las líneas 7 a la 12 se define al modelo como un modelo celular de 10x10x3 celdas, es decir 3 planos de 10x10 celdas con demora de transporte y un tiempo de demora por defecto de 100 milisegundos. En las líneas 13 a 21 se define la vecindad. En este modelo es muy particular. En primer lugar se debe definir a las 8 celdas vecinas en el plano 0, por otro lado se necesita los 9 vecinos directamente debajo de las celdas que conforman la vecindad y la celda que se esta analizando, para que se pueda analizar la temperatura del área y por ultimo se necesita solamente la celda que esta en el plano 2 para que indique en que paso de la simulación se esta.

En las líneas 23 a 25 se definen las conexiones de los dos generadores de calor con las celdas del plano 1. La línea 33 contiene la primer regla, esta será evaluada cuando se este analizando una celda que este en el plano 0, el valor de la celda ubicada en el plano 2 es igual a cero, la celda que se esta evaluando debe tener el valor 1, es decir, que el misil se dirige hacia el norte y por ultimo la diferencia de temperatura entre los dos vecinos que están a la derecha e izquierda de la celda en análisis. Si esta diferencia es menor a cero se deberá girar al misil hacia la derecha tomando el nuevo valor 2, que indica que el misil se dirige hacia el este. En la línea 34 se analiza lo mismo pero si la diferencia de temperatura es mayor a cero deberá girar hacia la izquierda tomando el valor 4 que indica que el misil se dirige hacia el oeste. En la línea 35 y 36 se evalúa si el misil tiene dirección hacia el este con diferencia de temperatura entre los 2 vecinos que posee a sus costados es menor a cero girara hacia el sur y sino girara hacia el norte. En las líneas 37, 38, 39 y 40 se analiza lo mismo con la diferencia en la dirección del misil (ver Figura 25). En las líneas 42 a 45 se avanza una celda hacia delante. La nueva celda que ocupara el misil dependerá de la dirección del mismo.



(a) (b) (c)

Figura 25 – (a) El misil se dirige hacia el este las celdas que se encuentran en sus laterales definirán la nueva dirección. (b) Como la celda que se encontraba a la derecha del misil tenia mayor temperatura el misil giró y avanzo hasta esa posición teniendo ahora una dirección hacia el norte. (c) Se muestran las nuevas celdas que definirán la nueva dirección del misil.

En las reglas de las líneas 47 y 49 se adelanta el misil y se coloca en cero el valor de la celda donde estaba ubicado el misil en el paso anterior. La regla de la línea 51 se evalúa solo en las celdas del plano 1. El sentido de esta regla es disipar el calor a toda el plano. Cada celda del plano 1 toma un nuevo valor debido a que se promedian todos los valores de los vecinos. En las líneas 53 y 54 se incrementa el valor de las celdas del plano 2 que eran utilizadas como semáforos para realizar determinada operación.

4.6 Robots

El modelo consiste en un piso de una planta en la cual robots semiautónomos acarrean materiales y materia elaborada siguiendo rutas o caminos específicos. Estos caminos están prefijados y existen tramos donde se pueden cruzar con otras rutas de otros robots, es por esta razón que se debe tener en cuenta el análisis de colisiones.

Nuevamente se tiene un modelo multicomponentes: un modelo celular bidimensional de 40x70 celdas que representa la superficie de la planta industrial y cinco generadores de carga son los encargados de simular la entrada de carga a la planta.

Se tienen cinco rutas diferentes y los robots pueden ir en una de las siguientes direcciones: Norte, Sur, Este y Oeste. Los robots avanzan sobre cada ruta a distintas velocidades. La velocidad depende de la distancia al robot mas cercano: se reduce la velocidad en el caso de tener un robot cerca, para evitar choques. Además cada robot deberá tener en cuenta la posibilidad de colisionar con otro robot en una zona de cruce de rutas diferentes. Las fuentes proveen materiales que deben ser acarreados hacia los destinos, donde serán consumidos. La carga tiene asociada una demorando aleatorio entre 20 y 90 segundos dependiendo el material a transportar, luego recorre la ruta que tiene asignada hasta dejar la carga en el destino tomándose también tiempo para la descarga. Una vez que la carga es depositada en el destino, el robot sale de la planta para volver a comenzar su trabajo por la entrada, por lo tanto dentro de una misma ruta el sentido de los robots es único. Por ultimo la llegada de carga es manejada de manera aleatoria.

Cada robot tendrá asignado un numero de ruta comenzando con la ruta 1 a hasta la ruta 5, como así también la dirección en el sentido que se esta moviendo. Luego en cada celda el primer dígito del valor de la misma representa la ruta y el segundo dígito la dirección. Por lo tanto si una celda tiene el valor 43 significa que en esa celda existe un robot perteneciente a la ruta 4 con dirección Sur. Luego los valores para representar la dirección serán: 1 - en dirección Norte, 2 - en dirección Este, 3 - en dirección Sur y 4 - en dirección Oeste. En el caso de las rutas simplemente se utiliza el número que le corresponda. El valor cero en la celda significa la no presencia de un robot.

Debido a que es un modelo Cell-DEVS acoplado bidimensional, puede definirse como:

$$CC = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, \eta, N, C, B, Z, select >$$

donde:

Xlist = {(5,69), (10,69), (20,69), (39,10), (0,30)} corresponde a las celdas donde existe acoplamiento interno

Ylist = {∅} debido a que no tiene acoplamiento externo

$I = < P^x, P^y >$ donde:

P^x = { <X(5,69), Real>, <X(10,69), Real>, <X(20,69), Real>, <X(39,10), Real>, <X(0,30), Real>}

$P^y = \{\varnothing\}$

$n = 2$

$\{t_1=70, t_2=40\}$

$\eta = 17$

$X \subseteq \Re$

$Y \subseteq \Re$

N = {(-1,-1), (-1,0), (-1,1), (0,-1), (0,0) , (0,1), (1,-1) (1,0), (1,1), (0,2), (0,3), (0,-2), (0,-3), (2,0), (3,0), (-2,0), (-3,0)}

C = $\{C_{k1, k2, ..., kn} / k_i \in [0,t_i] \forall i \in N, i \in [1,n]\}$, es el conjunto de celdas atómicas que conforman al modelo acoplado, donde:

$C_{ij} = < X, Y, I, demora, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$

donde

I = <17,0,P^x,P^y>

P^x = { (X_1,real), (X_2,real), (X_3,real), (X_4,real), (X_5,real), (X_6,real), (X_7,real), (X_8,real), (X_9,real), (X_{10},real), (X_{11},real), (X_{12},real), (X_{13},real), (X_{14},real), (X_{15},real), (X_{16},real), (X_{17},real)}

P^y = { (Y_1,real), (Y_2,real), (Y_3,real), (Y_4,real), (Y_5,real), (Y_6,real), (Y_7,real), (Y_8,real), (Y_9,real), (Y_{10},real), (Y_{11},real), (Y_{12},real), (Y_{13},real), (Y_{14},real), (Y_{15},real), (Y_{16},real), (Y_{17},real)}

demora = transporte

S:

Variables descriptivas

$s \in \Re$

Parámetros:

movtmp = demora relacionada con la velocidad de los robots. La misma es de 1000 milisegundos

N = {(-1,-1), (-1,0), (-1,1), (0,-1), (0,0) , (0,1), (1,-1) (1,0), (1,1), (0,2), (0,3), (0,-2), (0,-3), (2,0), (3,0), (-2,0), (-3,0)}

d = *movtmp*

La definición de las rutas puede verse en la Figura 26. En la Figura 27 se presenta la especificación del modelo en N-CD++.

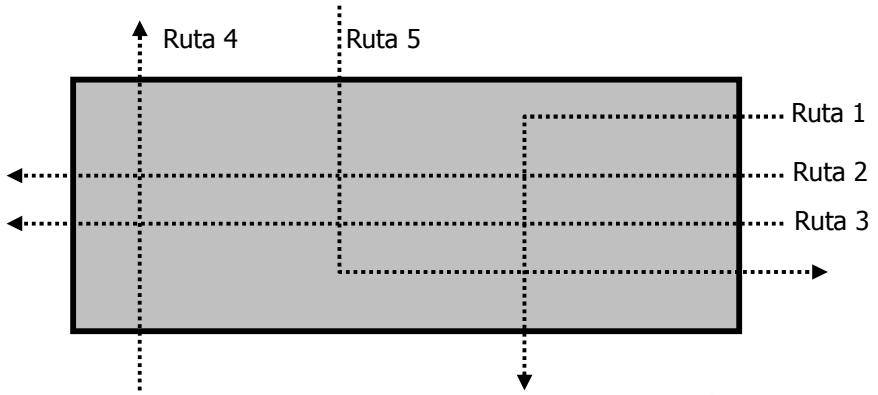


Figura 26 – Trazado de las rutas con el sentido de circulación de los robots.

```
01 [top]
02 components : Planta Fuente1@Generator Fuente2@Generator Fuente3@Generator
   Fuente4@Generator Fuente5@Generator
03 link : out@Fuente1 in1@Planta
04 link : out@Fuente2 in2@Planta
05 link : out@Fuente3 in3@Planta
06 link : out@Fuente4 in4@Planta
07 link : out@Fuente5 in5@Planta
08 [Fuente1]
09 distribution : poisson
10 mean : 3
11 initial : 1
12 increment : 0
13 [Fuente2]
14 distribution : poisson
15 mean : 4
16 initial : 1
17 increment : 0
18 [Fuente3]
19 distribution : poisson
20 mean : 7
21 initial : 1
22 increment : 0
23 [Fuente4]
24 distribution : poisson
25 mean : 9
26 initial : 1
27 increment : 0
28 [Fuente5]
29 distribution : poisson
30 mean : 3
31 initial : 1
32 increment : 0
33 [Planta]
34 type : cell
35 dim : (70,40)
36 delay : transport
37 defaultDelayTime : 1000
38 border : nowrapped
39 neighbors :                                          Planta(-3,0)
40 neighbors :                                          Planta(-2,0)
41 neighbors :                              Planta(-1,-1) Planta(-1,0) Planta(-1,1)
42 neighbors : Planta(0,-3)  Planta(0,-2)Planta(0,-1)  Planta(0,0)  Planta(0,1)
   Planta(0,2) Planta(0,3)
43 neighbors :                              Planta(1,-1)  Planta(1,0)  Planta(1,1)
44 neighbors :                                          Planta(2,0)
45 neighbors :                                          Planta(3,0)
```

```
initialValue : 0
initialCellsValue : Robots.val
in : in1 in2 in3 in4 in5
link : in1 in@Planta(5,69)
link : in2 in@Planta(10,69)
link : in3 in@Planta(20,69)
link : in4 in@Planta(39,10)
link : in5 in@Planta(0,30)
localtransition : RobotsMov
portInTransition : in@Planta(5,69) GenRobot1
portInTransition : in@Planta(10,69) GenRobot2
portInTransition : in@Planta(20,69) GenRobot3
portInTransition : in@Planta(39,10) GenRobot4
portInTransition : in@Planta(0,30) GenRobot5
[RobotsMov]
%------------------------- Ruta Uno ------------------------------------
rule : 13 {if((2,0)=13 or (1,0)=13,2000,1000)} { (0,0)=0 and (-1,0)=13 and
   (0,1)=0 and (0,-1)=0 and (1,0)!=?}
rule : {(1320 + randInt(70))} 1000 { (0,0)=0 and (-1,0)=13 and (1,0)=?}
rule : {((0,0) - 1)} 1000 {(0,0)>1300 and (0,0)<=1399}
rule : 0 1000 {(0,0)=1300}
rule : 0 {if((3,0)=13 or (2,0)=13,2000,1000)} {(0,0) = 13 and (1,0)=0 and
   (1,1)=0 and (1,-1)=0}
rule : 14 {if((0,-2)=14 or (0,-1)=14,2000,1000)} { (0,0)=0 and (0,1)=14 and
   (0,-1)!=? and cellpos(1)>50}
rule : {(1420 + randInt(70))} 100 { (0,0)=0 and (0,1)=14 and (0,-1)=?}
rule : {((0,0) - 1)} 1000 {(0,0)>1400 and (0,0)<=1499}
rule : 0 1000 {(0,0)=1400}
rule : 0 {if((0,-3)=14 or (0,-2)=14,2000,1000)} {(0,0)=14}
rule : 13 1000 { (0,0)=0 and (0,1)=14 and (0,-1)!=? and cellpos(1)=50}
%------------------------- Ruta Dos ------------------------------------
rule : 24 {if((0,-2)=24 or (0,-1)=24,2000,1000)} { (0,0)=0 and (0,1)=24 and
   (0,-1)!=?}
rule : {(2420 + randInt(70))} 1000 { (0,0)=0 and (0,1)=24 and (0,-1)=?}
rule : {((0,0) - 1)} 1000 {(0,0)>2400 and (0,0)<=2499}
rule : 0 1000 {(0,0)=2400}
rule : 0 {if((0,-3)=24 or (0,-2)=24,2000,1000)} {(0,0)=24}
%------------------------- Ruta Tres-----------------------------------
rule : 34 {if((0,-2)=34 or (0,-1)=34,2000,1000)} { (0,0)=0 and (0,1)=34 and
   (0,-1)!=?}
rule : {(3420 + randInt(70))} 1000 { (0,0)=0 and (0,1)=34 and (0,-1)=?}
rule : {((0,0) - 1)} 1000 {(0,0)>3400 and (0,0)<=3499}
rule : 0 1000 {(0,0)=3400}
rule : 0 {if((0,-3)=34 or (0,-2)=34,2000,1000)} {(0,0)=34}
%------------------------- Ruta Cuatro ---------------------------------
rule : 41 {if((-2,0)=41 or (-1,0)=41,2000,1000)} { (0,0)=0 and (1,0)=41 and
   (0,1)=0 and (0,-1)=0 and (-1,0)!=?}
rule : {(4120 + randInt(70))} 1000 { (0,0)=0 and (1,0)=41 and (-1,0)=?}
rule : {((0,0) - 1)} 1000 {(0,0)>4100 and (0,0)<=4199}
rule : 0 1000 {(0,0)=4100}
rule : 0 {if((-3,0)=41 or (-2,0)=41,2000,1000)} {(0,0)=41 and (-1,-1)=0 and
   (-1,0)=0 and (-1,1)=0}
%------------------------- Ruta Cinco-----------------------------------
rule : 53 {if((2,0)=53 or (1,0)=53,2000,1000)} { (0,0)=0 and (-1,0)=53 and
   (0,1)=0 and (0,-1)=0 and (1,0)!=? and cellpos(0)<30}
rule : {(5320 + randInt(70))} 1000 { (0,0)=0 and (-1,0)=53 and (1,0)=?}
rule : {((0,0) - 1)} 1000 {(0,0)>5300 and (0,0)<=5399}
rule : 0 1000 {(0,0)=5300}
rule : 0 {if((3,0)=53 or (2,0)=53,2000,1000)} {(0,0) = 53 and (1,0)=0 and
   (1,1)=0 and (1,-1)=0}
rule : 52 {if((0,2)=52 or (0,1)=52,2000,1000)} { (0,0)=0 and (0,-1)=52 and
   (0,1)!=?}
rule : {(5220 + randInt(70))} 1000 { (0,0)=0 and (0,-1)=52 and (0,1)=?}
rule : {((0,0) - 1)} 1000 {(0,0)>5200 and (0,0)<=5299}
rule : 0 100 {(0,0)=5200}
rule : 0 {if((0,3)=52 or (0,2)=52,2000,1000)} {(0,0) = 52}
```

```
102 rule : 52 1000 { (0,0)=0 and (-1,0)=53 and (0,1)=0 and (0,-1)=0 and (1,0)!=?
    and cellpos(0)=30}
103 rule : {(0,0)} 1000 { t }
104
105 [GenRobot1]
106 rule : 14 1000 { t }
107 [GenRobot2]
108 rule : 24 1000 { t }
109 [GenRobot3]
110 rule : 34 1000 { t }
111 [GenRobot4]
112 rule : 41 1000 { t }
113 [GenRobot5]
114 rule : 53 1000 { t }
```

Figura 27 – Especificación del modelo robots.

En la línea 2 se definen los componentes, los componentes Fuente1, Fuente2, Fuente3, Fuente4 y Fuente5 son componentes derivados de la clase Generador. Este modelo DEVS ya viene incorporado dentro de la herramienta y permite generar, en un intervalo aleatorio predefinido, valores particulares. De la línea 8 a la 32 tenemos la definición del intervalo de tiempo que será utilizado por cada generador para generar el arribo de materiales. Todo los casos utilizan una distribución Poisson, variando solamente la media.

En la línea 33 comienza la definición del componente celular. Debido a que es un modelo acoplado se deben definir los puertos de entradas y asociar a estos a celdas especificas. Por ejemplo en la línea 49 se conecta el puerto **in1**, definido como puerto de entrada en la línea 48, con la celda ubicada en la posición (5,69). En las líneas 55 a 59 se asigna celda conectada a puertos de entrada un comportamiento especifico, es decir que cada vez que la celda recibe un evento por el puerto se utilizaran reglas definidas en una sección en particular, por ejemplo en la línea 56, si la celda recibe un evento por el puerto de entrada conectado al generador utilizara las reglas definidas en la sección **GenRobot2**, dicha sección no es la misma que rige el comportamiento de las celdas del modelo.

En la línea 62 a la 104 se definen las reglas para el movimiento y detección de colisiones de los robots para cada ruta específica. Para la ruta1 se tiene la carga de materiales en la parte superior derecha (ver Figura 26). El robot avanza hasta llegar a una posición especifica de la planta, gira a la izquierda y se mueve en dirección Sur. Las reglas para esta ruta se encuentran desde la línea 62 a 72. La regla de la línea 62 es la que determina la velocidad de avance y verifica si existe otro robot delante en la misma ruta, si esto es verdadero la demora de avance es mayor a que si esto no fuera cierto. En la línea 63 verifica si existe algún robot que tenga que cargar material para acarrear y aleatoriamente asigna un tiempo de demora de carga, este puede variar entre 20 y 90 segundo, se asigna un valor a la celda que se decrementará cada un segundo, una vez que se cumple este lapso de tiempo el robot comienza su recorrida por la ruta asignada. La regla de la línea 64 es la encargada de decrementar el contador de carga. La línea 65 coloca a la celda en valor cero una vez que el contador de descarga llega cero. La regla de la línea 66 es la encargada de controlar el movimiento y de avanzar solo si el robot no colisionara con otro robot, ya sea de la misma ruta como de las rutas que se cruzan. En la línea 67 se encuentra la regla que analiza la posición del robot para que, si alcanza la columna 50, gire a la izquierda y tome la misma ruta per en dirección Sur. Las reglas de las líneas 68 a 72 son las encargadas de la descarga. Como en la línea 63, se asigna un tiempo aleatorio para descarga (demora por descarga) y una vez cumplido ese tiempo se hace salir al robot de la planta industrial para que, cuando sea solicitada, vuelva a entrar para transportar mas materiales.

Las reglas que siguen actúan de la misma manera que las de la ruta1, cambiando solamente las direcciones de los robots y las rutas.

En las líneas 105 a 114, se tienen las secciones que contienen las reglas que serán evaluadas cuando arribe material a cada puerto de entrada a la planta industrial. Por ejemplo en la línea 106 se encuentra la regla que será analizada cuando llegue material al puerto de entrada de la ruta1. Lo que hace esa regla es asignar a la celda un robot con dirección Oeste, dado que se necesitara un robot para transportar el material que arribo y la ruta es en esa dirección.

5. Aplicaciones complejas

En esta sección se analizaran implementaciones de modelos complejos. Los mismos son modelos biológicos y naturales.

5.1 Hormigas

Se comienza con la implementación de un modelo de hormigas moviéndose por el suelo. Las hormigas comen pasto en dos pasos: primero, comen la parte de arriba del pasto dejando la raíz para luego en otro paso comerla. El modelo es un modelo celular tridimensional (la necesidad de varios planos es debido a que se necesita tener información adicional para la detección de colisiones) .

Cuando una hormiga encuentra pasto come la parte superior y rota 90º hacia la derecha. Cuando solo queda la raíz del pasto, y una hormiga se la come, rota a 90º hacia la izquierda. Por ultimo si la hormiga no encuentra pasto sigue siempre caminando en la misma dirección. Otro aspecto que tiene en cuenta este modelo es el crecimiento del pasto, esto hace que las hormigas tengan siempre algo para comer.

El modelo usa dos planos, en el plano 0 se modela el comportamiento de las hormigas y del pasto y al plano 1 se lo utilizara para determinar aspectos tales como análisis de posibles colisiones o si el pasto puede crecer. Se usan 4 estados para representar la dirección de las hormigas, que a su vez fueron combinados con el estado del pasto. Esto hace que cada valor de celda contenga 2 posiciones la primera para indicar el estado del suelo y la segunda posición indica la presencia y dirección de la hormiga. Luego se tendrán los siguientes valores:

Para las hormigas
0: Indica que no hay presencia en la celda de una hormiga
1 y 5: Indica que hay una hormiga en la celda y se dirige en dirección norte
2 y 6: La celda contiene una hormiga y se dirige en dirección este.
3 y 7: La celda contiene una hormiga y se dirige en dirección sur.
4 y 8: La celda contiene una hormiga y se dirige en dirección este.

Se los valores 5, 6, 7 y 8 son valores auxiliares utilizados para saber donde estará la hormiga en el próximo paso y poder detectar colisiones.

Para el suelo
1: Indica que existe pasto y no fue comido nunca
2: La celda solo contiene la raíz del pasto
3 y 4: La celda no contiene pasto.

Por ejemplo si una celda contiene el valor 13 significa que en esa posición existe pasto sin comer y que la hormiga se dirige en dirección sur.

```
01      [top]
02      components : vants
03
04      [vants]
05      type : cell
06      dim : (10,10,2)
07      delay : transport
08      defaultDelayTime  : 100
09      border : wrapped
10
11      neighbors :                              vants(-2,0,0)
12      neighbors :               vants(-1,-1,0) vants(-1,0,0) vants(-1,1,0)
```

```
13      neighbors : vants(0,-2,0)  vants(0,-1,0)  vants(0,0,0)  vants(0,1,0)
vants(0,2,0)
14      neighbors :                vants(1,-1,0)  vants(1,0,0)  vants(1,1,0)
15      neighbors :                               vants(2,0,0)
16      neighbors :                               vants(0,0,1)
17
18      initialvalue : 1
19      initialCellsValue : vantsIII.val
20      localtransition : calculus
21
22      [calculus]
23      rule : 26 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=11 }
24      rule : 27 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=12 }
25      rule : 28 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=13 }
26      rule : 25 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=14 }
27
28      rule : 38 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=21 }
29      rule : 35 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=22 }
30      rule : 36 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=23 }
31      rule : 37 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=24 }
32
33      rule : 45 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=31 }
34      rule : 46 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=32 }
35      rule : 47 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=33 }
36      rule : 48 100 { cellpos(2)=0 and (0,0,1)=1 and (0,0,0)=34 }
37
38      rule : { if((0,0,0)=45,31,(trunc(((0,0,0)/10))*10) + 1) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=25 or (0,0,0)=35 or (0,0,0)=45) and
(fractional((-1,0,0)/10)*10)!=7 and ((fractional((-1,1,0)/10)*10)=8 or
(fractional((-2,0,0)/10)*10)=7 or (fractional((-1,-1,0)/10)*10)=6) }
39
40      rule : { if((0,0,0)=45,33,(trunc(((0,0,0)/10))*10) + 3) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=25 or (0,0,0)=35 or (0,0,0)=45) and
(fractional((-1,0,0)/10)*10)=7 }
41
42      rule : { if((0,0,0)=48,34,(trunc(((0,0,0)/10))*10) + 4) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=28 or (0,0,0)=38 or (0,0,0)=48) and
(fractional((0,-1,0)/10)*10)!=6 and (((fractional((0,-2,0)/10)*10)=6 and
(fractional((-1,-1,0)/10)*10)=0 and (fractional((1,-1,0)/10)*10)=0 ) or
((fractional((-1,-1,0)/10)*10)=7 and (fractional((0,-2,0)/10)*10)=6 and
(fractional((1,-1,0)/10)*10)=0)) }
43      rule : { if((0,0,0)=48,32,(trunc(((0,0,0)/10))*10) + 2) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=28 or (0,0,0)=38 or (0,0,0)=48) and
(fractional((0,-1,0)/10)*10)=6 }
44
45      rule : { if((0,0,0)=46,32,(trunc(((0,0,0)/10))*10) + 2) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=26 or (0,0,0)=36 or (0,0,0)=46) and
(fractional((0,1,0)/10)*10)!=8 and (((fractional((-1,1,0)/10)*10)=7 and
(fractional((1,1,0)/10)*10)=0 and (fractional((0,2,0)/10)*10)=0) or
((fractional((0,2,0)/10)*10)=8 and (fractional((1,1,0)/10)*10)=5 and
(fractional((-1,1,0)/10)*10)=0) or ((fractional((0,2,0)/10)*10)=8 and
(fractional((1,1,0)/10)*10)=5 and (fractional((-1,1,0)/10)*10)=7)) }
46      rule : { if((0,0,0)=46,34,(trunc(((0,0,0)/10))*10) + 4) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=26 or (0,0,0)=36 or (0,0,0)=46) and
(fractional((0,1,0)/10)*10)=8 }
47
48      rule : { if((0,0,0)=47,33,(trunc(((0,0,0)/10))*10) + 3) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=27 or (0,0,0)=37 or (0,0,0)=47) and
(fractional((1,0,0)/10)*10)!=5 and (((fractional((1,-1,0)/10)*10)=6 and
(fractional((1,1,0)/10)*10)=8 and (fractional((2,0,0)/10)*10)=0) or
((fractional((1,1,0)/10)*10)=8 and (fractional((2,0,0)/10)*10)=5 and
(fractional((1,-1,0)/10)*10)=0) or ((fractional((1,1,0)/10)*10)=8 and
(fractional((2,0,0)/10)*10)=0 and (fractional((1,-1,0)/10)*10)=0) or
((fractional((1,-1,0)/10)*10)=6 and (fractional((2,0,0)/10)*10)=5 and
(fractional((1,1,0)/10)*10)=0) or ((fractional((1,-1,0)/10)*10)=6 and
(fractional((2,0,0)/10)*10)=5 and (fractional((1,1,0)/10)*10)=8)) }
```

```
49      rule : { if((0,0,0)=47,31,(trunc(((0,0,0)/10))*10) + 1) } 100 {
cellpos(2)=0 and (0,0,1)=2 and ((0,0,0)=27 or (0,0,0)=37 or (0,0,0)=47) and
(fractional((1,0,0)/10)*10)=5}
50
51      rule : { (trunc(((0,0,0)/10))*10) + 1 } 100 {cellpos(2)=0 and (0,0,1)=3
and ((0,0,0)=25 or (0,0,0)=35 or (0,0,0)=45) and ((fractional((-1,1,0)/10)*10)=8
or (fractional((-1,-1,0)/10)*10)=6 or (fractional((-1,0,0)/10)*10)!=0)}
52      rule : { (trunc(((0,0,0)/10))*10) + 2 } 100 {cellpos(2)=0 and (0,0,1)=3
and ((0,0,0)=26 or (0,0,0)=36 or (0,0,0)=46) and ((fractional((-1,1,0)/10)*10)=7
or (fractional((1,1,0)/10)*10)=5 or (fractional((0,1,0)/10)*10)!=0)}
53      rule : { (trunc(((0,0,0)/10))*10) + 3 } 100 {cellpos(2)=0 and (0,0,1)=3
and ((0,0,0)=27 or (0,0,0)=37 or (0,0,0)=47) and ((fractional((1,1,0)/10)*10)=8
or (fractional((1,-1,0)/10)*10)=6 or (fractional((2,0,0)/10)*10)=5 or
(fractional((1,0,0)/10)*10)!=0)}
54      rule : { (trunc(((0,0,0)/10))*10) + 4 } 100 {cellpos(2)=0 and (0,0,1)=3
and ((0,0,0)=28 or (0,0,0)=38 or (0,0,0)=48) and ((fractional((-1,-1,0)/10)*10)=7
or (fractional((1,-1,0)/10)*10)=5 or (fractional((0,-2,0)/10)*10)=6 or
(fractional((0,-1,0)/10)*10)!=0)}
55
56      rule : { (trunc(((0,0,0)/10))*10) + 1 } 100 {cellpos(2)=0 and (0,0,1)=4
and ((1,0,0)=25 or (1,0,0)=35 or (1,0,0)=45)}
57      rule : { (trunc(((0,0,0)/10))*10) + 2 } 100 {cellpos(2)=0 and (0,0,1)=4
and ((0,-1,0)=26 or (0,-1,0)=36 or (0,-1,0)=46)}
58      rule : { (trunc(((0,0,0)/10))*10) + 3 } 100 {cellpos(2)=0 and (0,0,1)=4
and ((-1,0,0)=27 or (-1,0,0)=37 or (-1,0,0)=47)}
59      rule : { (trunc(((0,0,0)/10))*10) + 4 } 100 {cellpos(2)=0 and (0,0,1)=4
and ((0,1,0)=28 or (0,1,0)=38 or (0,1,0)=48)}
60
61      rule : 20 100 {cellpos(2)=0 and (0,0,1)=5 and (0,0,0)>=25 and (0,0,0)<=28
}
62      rule : 30 100 {cellpos(2)=0 and (0,0,1)=5 and (0,0,0)>=35 and (0,0,0)<=38
}
63      rule : 30 100 {cellpos(2)=0 and (0,0,1)=5 and (0,0,0)>=45 and (0,0,0)<=48}
64      rule : {30 + fractional((0,0,0)/10)*10} 100 {cellpos(2)=0 and (0,0,1)=5
and (0,0,0)>=41 and (0,0,0)<=44}
65
66      rule : 20 2973 {cellpos(2)=0 and (0,0,0)=30 and statecount(20)>=4 and
(statecount(10)+statecount(20)+statecount(30)>12)}
67      rule : 10 2677 {cellpos(2)=0 and (0,0,0)=20 and statecount(10)>=3 and
(statecount(10)+statecount(20)+statecount(30)>12)}
68      rule : { (0,0,0) + 1 } 100 {cellpos(2)=1 and (0,0,0)<7}
69      rule : 1 100 {cellpos(2)=1 and (0,0,0)>=7}
70
71      rule : { (0,0,0) } 100 { t }
```

Figura 28 – Implementación del modelo de Hormigas

En la Figura 28 se detalla la implementación del modelo. En las líneas 1 a 9 se define el único componente del modelo, como un modelo celular de 10x10x2 celdas, es decir dos planos de 10x10 celdas. Se asigna como tipo de demora por defecto a la demora de transporte. Los bordes son toroidales o wrapped. De la línea 11 a la 16 se define la vecindad. Como se ve en la Figura 29 se agregaron una mayor cantidad de vecinos que son utilizados para detectar las colisiones.

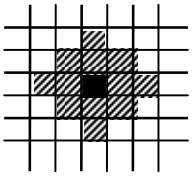


Figura 29 – Vecindad en el modelo Hormigas

En las líneas 18 a 20 se define los valores iniciales del modelo y el nombre de la sección que contendrá las reglas de comportamiento de las celdas.

En la línea 22 comienza la especificación de las reglas. De las líneas 23 a 36 se definen las reglas que serán las que determinen la nueva posición de las hormigas y a su vez el nuevo estado del pasto. Por ejemplo si se analiza la línea 23, se ve que esta se evaluará si se esta analizando una celda que: esta en el plano 0, el valor de la misma celda pero en el plano 1 tiene el valor 1 (este valor indica que se debe analizar el movimiento de las hormigas y estado del pasto) y contiene a una hormiga que se dirige hacia el norte sobre una superficie de pasto (que no fue comido anteriormente) debe tomar el nuevo valor 26 que significa que la hormiga comió parte del pasto (valor 2) y se dirigía hacia el norte (valor 6).

En las líneas 28 a 31 ocurre lo mismo pero en este caso solo había quedaba la raíz del pasto, dejando al suelo sin ninguna cantidad de pasto. De la línea 32 a 36 se evalúan cuando existen hormigas que van caminando por lugares donde no hay nada de pasto.

El siguiente conjunto de reglas analiza la posibilidad de colisión entre hormigas. Como cada hormiga solo puede tener cuatro direcciones posibles se tiene que analizar en cada dirección todas las posibilidades de colisión con otra hormiga. En caso de que se detecte una colisión se debe definir deterministicamente que hacer, es decir, que acciones se deberán tomar con cada una de las hormigas que están por colisionar. En la línea 38 se encuentra la primera de las reglas que analizan las posibilidades de colisión. Como se ve en la Figura 30 solo se tendrá en cuenta la posibilidad que existan hormigas que vengan del oeste o del este pero que la celda que esta frente no contenga a ninguna hormiga con dirección sur. También la regla se evaluara si se esta en una celda perteneciente al plano 0, si la misma celda pero del plano 1 tiene el valor 2, si en la celda existe una hormiga con dirección hacia el norte, en la celda que se encuentra delante de la hormiga no debe haber ninguna hormiga con dirección sur, si existe una hormiga con dirección este del lado izquierdo superior o si existe una hormiga con dirección oeste del lado superior derecho. El resultado de evaluar esta regla es que la hormiga que viene con dirección norte seguirá su camino, y las otras posibles hormigas deberán quedarse paradas.

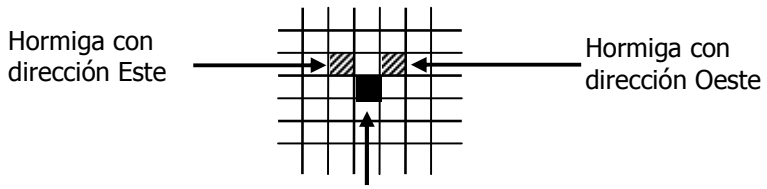


Figura 30 – Hormiga que puede colisionar con hormigas que vengan con dirección oeste o con dirección este

La regla de la línea 40 es evaluada si se está en una celda del plano 0, si en la celda en análisis existe una hormiga con dirección norte y si frente de ella hay una hormiga con dirección sur. En este caso el resultado de evaluar esta regla es que la hormiga que viene con dirección norte gire 180º para tomar la dirección sur debido que la hormiga que tiene delante no la deja seguir.

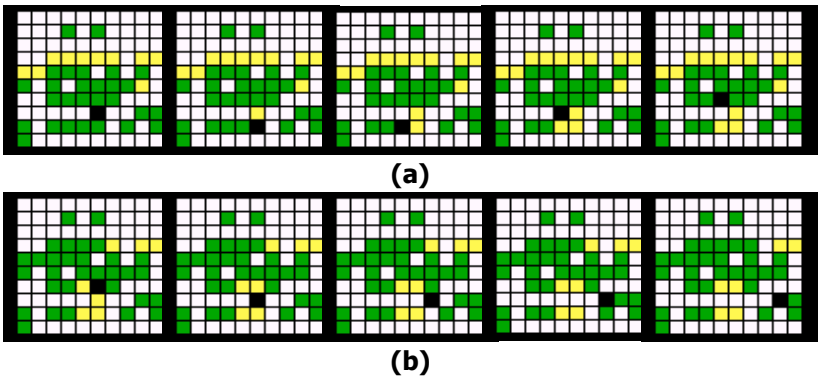
En las líneas 42 y 43 se analiza las mismas posibilidades pero de una hormiga con dirección este, en las líneas 45 y 46 si la hormiga tiene dirección oeste, en las líneas 48 y 49 si la hormiga tiene dirección sur. En todo estos casos si la hormiga se encuentra con una hormiga enfrente y con dirección opuesta gira 180, es decir toma la dirección contraria a la que venia.

Una vez que se analizaron estas posibilidades de colisión simples se debe hacer un nuevo análisis, debido a que puede pasar que una situación determinada no producía colisiones pero al evaluar alguna regla de las anteriores y cambiar la dirección de una hormiga se genera la posibilidad de colisión. La línea 51 es la primer regla de este tipo, y lo que estas reglas dejan como resultado de la evaluación es que las hormigas que queden en posibilidad de colisión nuevamente se detengan hasta la próxima evaluación de colisiones. Una vez que se analizaron las posibilidades de colisión, se deberán mover a aquellas hormigas que puedan. Esto se hace mediante la evaluación de las reglas de las líneas 56 a 64.

En las líneas 66 y 67 se encuentran las reglas que hacen crecer el pasto. El tiempo de demora en estas reglas es mucho más largo que el de movimiento de las hormigas. En la línea 66 se contempla la posibilidad de pasto solo cuando exista algún vecino con pasto, de alguna manera el pasto va creciendo

solo donde hay pasto cerca, y el resultado de evaluar esta regla será que la celda pasara a tener raíz de pasto. En la línea 67 se hace crecer la raíz de pasto a pasto listo para ser comido por alguna hormiga.

Por ultimo las líneas 67 y 68 son las que controlan los valores de las celdas del plano 1, utilizados para determinar que conjunto de reglas se deberán evaluar. El resultado de la ejecución del modelo se puede ver en la Figura 31.



(a)

(b)

Figura 31 – Ejecución del modelo Hormigas, las celdas blancas representan la ausencia de pasto, las celdas amarillas representan la raíz del pasto, las celdas verdes representan el pasto sin comer y por ultimo las celdas negras representan a las hormigas. (a) Los primeros cuatro pasos donde se puede ver como la hormiga cambia de dirección cada vez que come pasto. (b) La hormiga avanza hasta que encuentra pasto nuevamente

5.2 Watershed

En esta sección se toma como caso de estudio un fenómeno natural. Watershed (cuenca) son regiones naturales definidas por la forma de la superficie de la tierra, que acumula agua ya sea por lluvias, deshielo de montañas y desembocaduras de ríos, como muestra la Figura 32. Con el uso de sistemas geográficos computerizados tales como GIS (Geographic Information Systems) se pueden definir los limites del watershed, generando mapas con la topología del terreno con gran resolución y precisión.

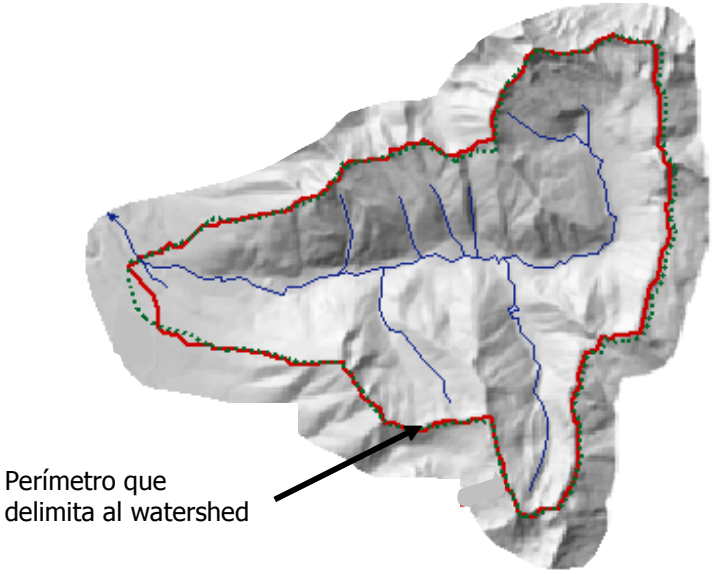


Figura 32 – Vista desde un sistema GIS de un Watershed

Un watershed esta compuesto por varias capas verticales: aire, vegetación, superficie del agua, tierra y rocas, luego el modelo puede ser dividido en pequeñas porciones de terreno de igual tamaño (celdas) donde a cada una de ellas se le puede aplicar un modelo conceptual hidrológico natural **[Moo96]** y analizar el comportamiento del flujo de agua, es decir, como varia la cantidad de agua en superficie y como la topología del terreno hace que la misma se concentre o acumule en determinados lugares.

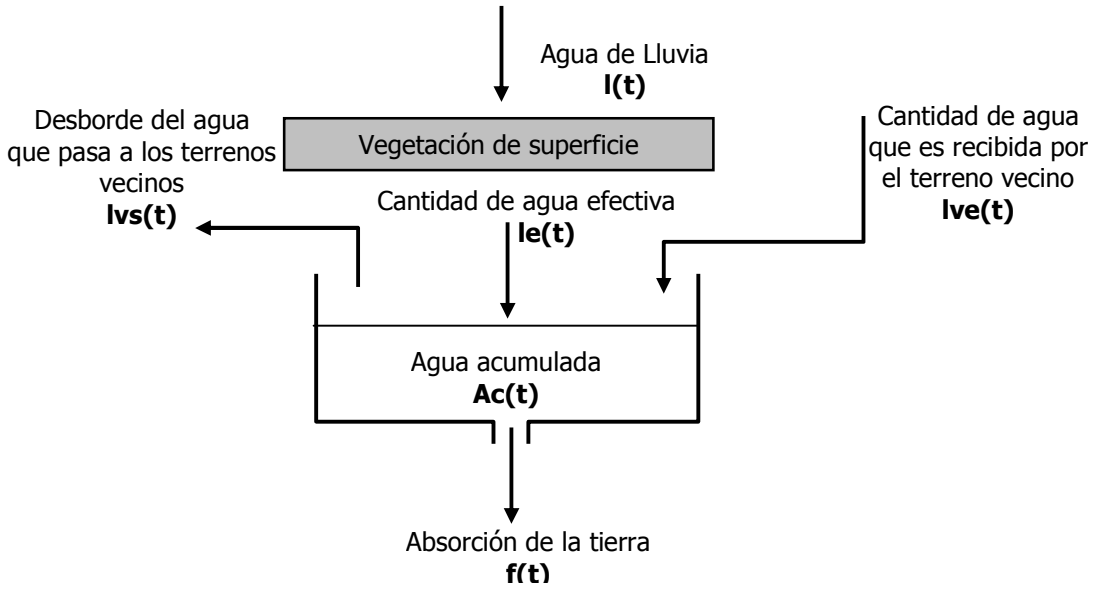


Figura 33 – Modelo hidrológico conceptual

La Figura 33 muestra el modelo hidrológico conceptual utilizado para la simulación. Al caer el agua de lluvia es interceptada por la vegetación existente en la superficie, el resto, la cantidad de agua efectiva es recibida por la superficie. También dependiendo de la topología del terreno, puede ingresar agua de las celdas vecinas, en este caso tomaremos como celdas vecinas a las cuatro celdas ubicadas al Norte, Sur, Este y Oeste. También puede pasar que el excedente de agua de la superficie pase, dependiendo de la inclinación o no del terreno a las celdas vecinas. Por ultimo parte del agua que alcanza la superficie se perderá por la filtración sobre la tierra y rocas. Luego sobre este modelo se puede definir la profundidad del agua a lo largo del tiempo en una celda mediante:

$$Ac(t) = \int_0^t (le(t) + \sum_{i=1}^{4} lve_i(t) - \sum_{i=1}^{4} lvs_i(t) - f(t))dt \quad (1)$$

$$lvs_i(t) = \frac{R_s C_i(t) Ac(t)}{D_i} \quad (2)$$

donde:

Ac = agua acumulada en una celda en milímetros
le = cantidad de agua efectiva
lve_i = cantidad de agua que vierten el vecino i a la celda
lvs_i = cantidad de agua que vierte la celda cuando supera su limite al vecino i
f = cantidad de agua que se filtra
R_s = parámetro que caracteriza la rugosidad de la superficie
D_i = la distancia que se encuentra entre la celda y el i-ésimo vecino
C_i = representa la inclinación del terreno con respecto al i-ésimo vecino y se calcula de la siguiente manera:

$$C(t) = \frac{Ac(t) + h - Ac_i(t) - h_i}{D_i} \quad (3)$$

donde:

Ac = agua acumulada en la celda
h = altitud de la celda
Ac_i = agua acumulada en la celda i
h_i = altitud de la celda i

Como se ve en la ecuación (1), la cantidad de agua acumulada a lo largo del tiempo depende de: la cantidad de agua de lluvia efectiva mas la cantidad de agua que es aportada (dependiendo de la inclinación del terreno) por las celdas vecinas menos el agua que es vertida, una vez superado el limite de capacidad de la celda, y menos la cantidad de agua que se filtra por la tierra y rocas.

En la ecuación (2) se define el aporte de agua que se vierte a los vecinos, el parámetro **C_i** es quien (ecuación (3)) determina la cantidad dependiendo la inclinación del terreno, es decir la diferencia de altitud entre la celda y sus vecinos.

La simulación se centra en la altura del agua alcanzada en cada celda a lo largo del tiempo con una intensidad de lluvia de 7,62 mm por hora, lluvia definida como de gran intensidad, constante a lo largo de todo el tiempo y en todo el terreno. Además se necesita incorporar topología del terreno, información que debe ser obtenida de algún sistema GIS o bien puede ser construida sobre información de mapas de suelo.

```
01      [top]
02      components : Watershed
03
04      [Watershed]
05      type : cell
06      dim : (30,30,2)
```

```
07      delay : transport
08      defaultDelayTime : 1000
09      border : nowrapped
10      neighbors :                    Watershed(-1,0,0)
11      neighbors : Watershed(0,-1,0) Watershed(0,0,0) Watershed(0,1,0)
12      neighbors :                    Watershed(1,0,0)
13
14      neighbors :                    Watershed(-1,0,1)
15      neighbors : Watershed(0,-1,1) Watershed(0,0,1) Watershed(0,1,1)
16      neighbors :                    Watershed(1,0,1)
17
18      initialValue : 0
19      initialCellsValue : topo_I.val
20      localtransition : Hydrology
21
22      [Hydrology]
23
24      rule : {0.0022 + (0,0,0) - if((((-1,0,0) != ?) and (((0,0,1) +
(0,0,0))>((-1,0,1) + (-1,0,0)))),(((((0,0,0) + (0,0,1) - (-1,0,0) - (-
1,0,1))/1000) * (0,0,0))/1000),0) - if((((1,0,0) != ?) and (((0,0,1) +
(0,0,0))>((1,0,1)  +  (1,0,0)))),(((((0,0,0) + (0,0,1) - (1,0,0) - (1,0,1))/1000)
* (0,0,0))/1000),0) - if((((0,-1,0) != ?) and (((0,0,1) + (0,0,0))>((0,-1,1)+(0,-
1,0)))),(((((0,0,0) + (0,0,1) - (0,-1,0) - (0,-1,1))/1000) * (0,0,0))/1000),0) -
if((((0,1,0) != ?) and (((0,0,1) + (0,0,0))>((0,1,1) + (0,1,0)))),(((((0,0,0) +
(0,0,1) - (0,1,0) - (0,1,1))/1000) * (0,0,0))/1000),0) + if((((-1,0,0) != ?) and
(((-1,0,1) + (-1,0,0))>((0,0,1) + (0,0,0)))),((((-1,0,0) + (-1,0,1) - (0,0,0) -
(0,0,1)) * (-1,0,0))/1000),0) + if((((1,0,0) != ?) and (((1,0,1) +
(1,0,0))>((0,0,1) + (0,0,0)))),((((1,0,0) + (1,0,1) - (0,0,0) - (0,0,1)) *
(1,0,0))/1000),0) + if((((0,-1,0) != ?) and (((0,-1,1) + (0,-1,0))>((0,0,1) +
(0,0,0)))),((((0,-1,0) + (0,-1,1) - (0,0,0) - (0,0,1)) * (0,-1,0))/1000),0) +
if((((0,1,0) != ?) and (((0,1,1) + (0,1,0))>((0,0,1) + (0,0,0)))),((((0,1,0) +
(0,1,1) - (0,0,0) - (0,0,1)) * (0,1,0))/1000),0) } 1000 { cellpos(2)=0 }
25      rule : { (0,0,0) } 1000 { t }
26
```

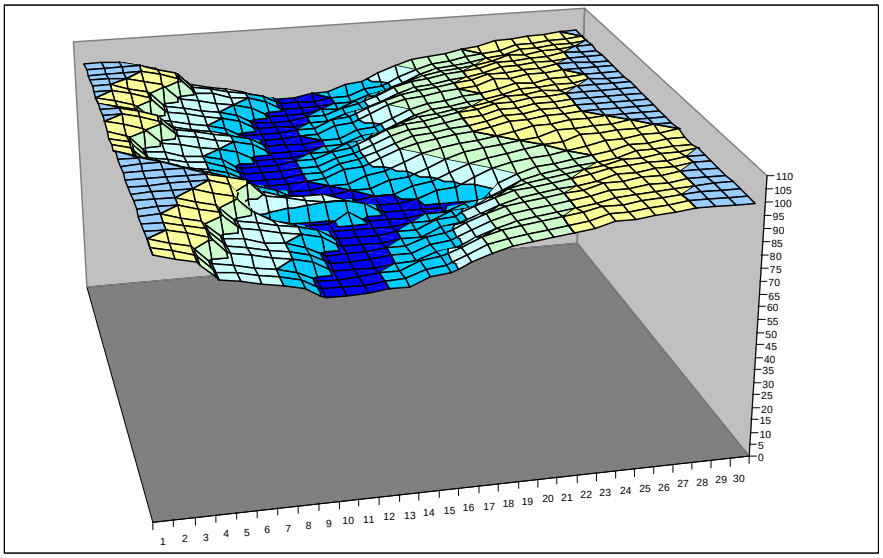
Figura 34 – Implementación del modelo Watershed

En la Figura 34 se presenta la especificación del modelo en el lenguaje utilizado por le **N-CD++**. En la línea 2 se define el único componente denominado Watershed. La definición del mismo comienza en la línea 4, como se menciono anteriormente, se definirá un modelo celular compuesto por dos planos, el plano 0 representara la cantidad de agua acumulada y el plano 1 representara las altitudes de cada celda con respecto al nivel del mar. Cada plano posee 30x30 celdas y cada una de ellas ocupa 1mx1m. En la línea 7 se define el tipo de demora como demora por transporte. La línea 8 define el tiempo de la demora por defecto en un segundo. La línea 9 define la forma de los bordes como de tipo nowrapped, es decir, que las celdas que están en los bordes no se conectan con las celdas de los bordes opuestos. De la línea 10 a 16 se define la vecindad, en cada uno de los planos se tomaron los cuatro vecinos ubicados al Norte, Sur, Este y Oeste debido a que el modelo solo considerara solamente intercambio de agua con estas celdas. En la línea 22 comienza la definición de las reglas de comportamiento del modelo. Solo existen 2 reglas, dado que el comportamiento es el mismo para todas las celdas que están el plano 0 y en el plano 1. La línea 24 define la cantidad de agua que contendrá la celda luego de 1 segundo, teniendo en cuenta: el agua que ya tenia anteriormente, mas el agua caída por la lluvia en ese lapso de tiempo, mas el agua que es cedida por los vecinos que ya superaron el nivel que pueden contener y menos el agua que, si es el caso, es cedida a los vecinos debido a que el nivel de agua máximo de la celda fue superado.

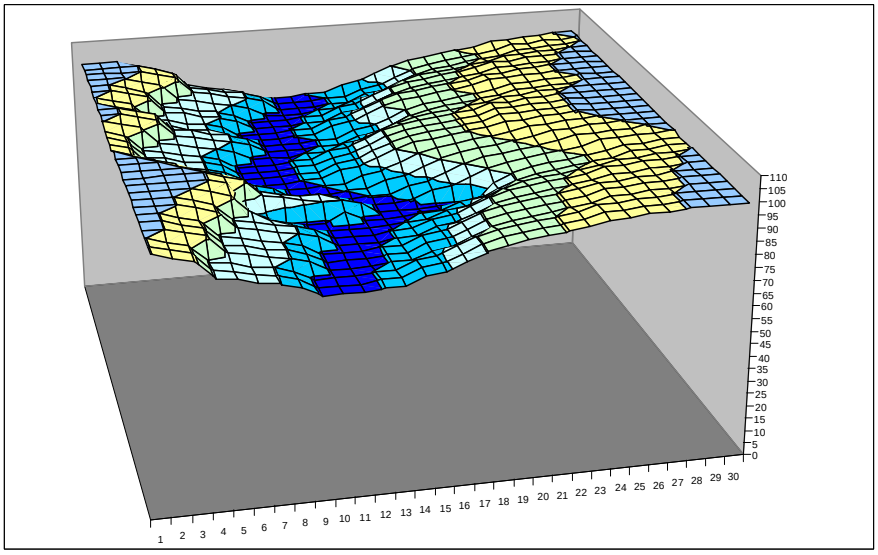
La cantidad de agua que es cedida como la cantidad de agua que es recibida de los vecinos depende de la cota máxima que soporta la celda y la inclinación del terreno. En la ecuación (3) se calcula de forma individual con cada celda vecina la inclinación del terreno. Si una celda vecina tiene mayor altitud que la celda en análisis, esta recibirá agua una vez superado el nivel de agua de la celda vecina. De la misma manera se calcula si la altitud de la celda en análisis es mayor que la de su vecino. La regla de la línea 24 comienza con el valor constante que representa la cantidad de agua efectiva que cae sobre la celda bajo una lluvia intensa (0.0022 mm/s). Este valor se suma a la cantidad de agua que ya tenia un

segundo antes, mas lo recibido por cada uno de los vecinos, menos lo que pasa a los vecinos. La regla de la línea 25 no modifica el estado de las celdas, debido a que siempre retorna el valor que tenia la celda.

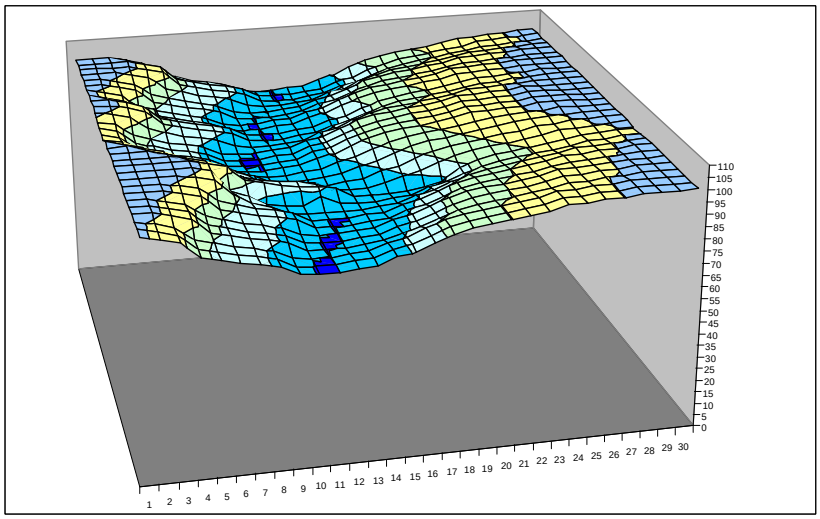
En la Figura 35 se pueden observar los resultados obtenidos luego de 45 minutos de lluvia intensa. En cada figura se aprecia los cambios de nivel del agua, y como afecta esto a la topología del terreno. Luego se puede concluir que transcurrido 45 minutos de lluvia intensa esta topología queda bajo el agua.



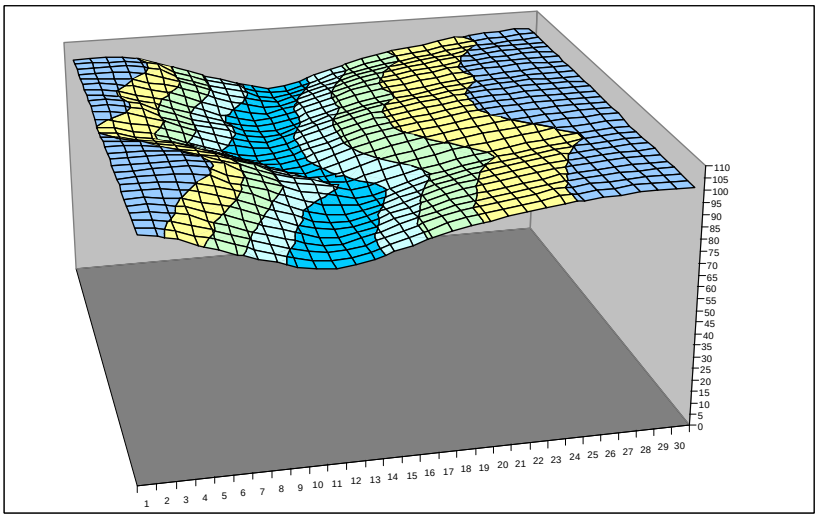
(a)



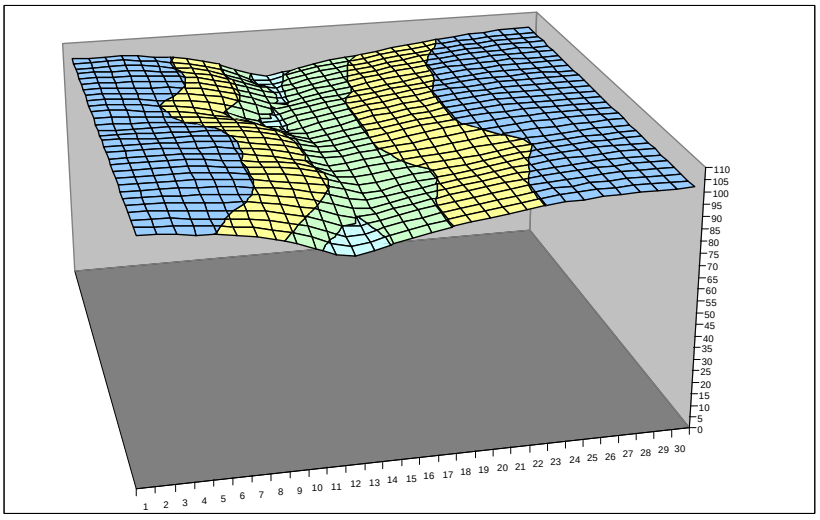
(b)


110 105 100 95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

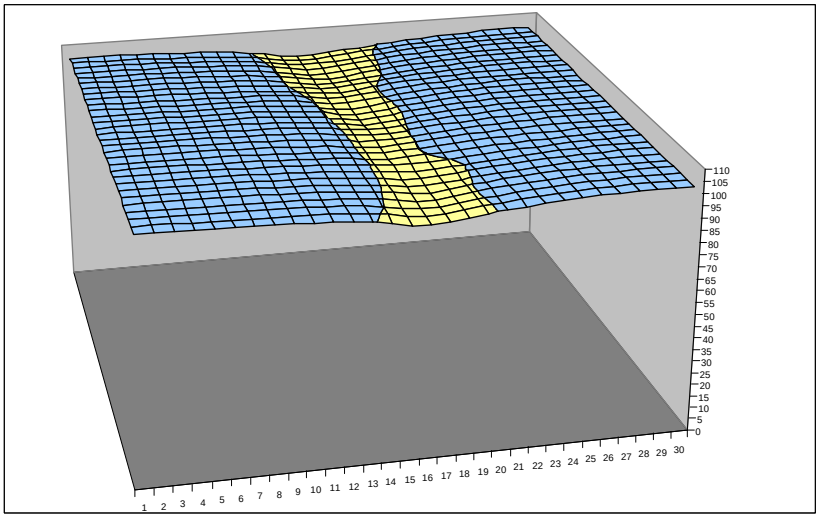
(c)


110 105 100 95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

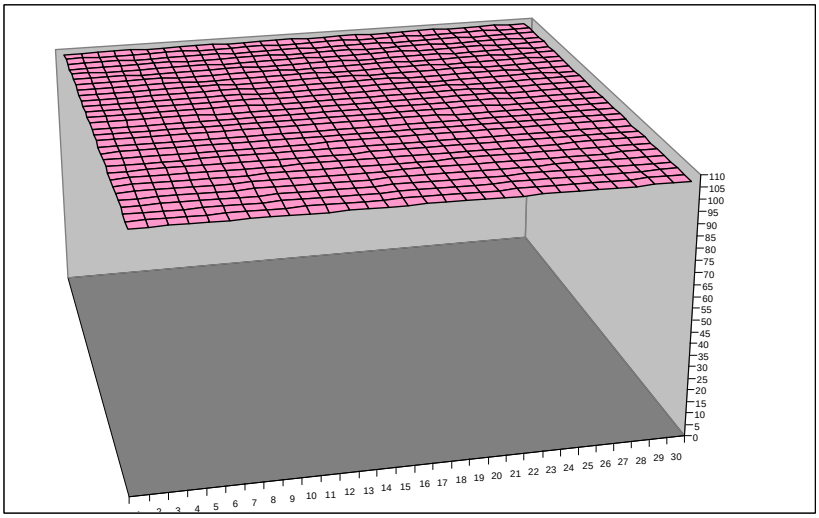
(d)


110 105 100 95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

(e)



(f)



(g)

Figura 35 – Resultados de la ejecución del modelo Watershed. (a) Topología del terreno. (b) Lluvia durante 5 minutos. (c) Luego de 10 minutos. (d) Luego de 15 minutos. (e) luego de 20 minutos. (f) Luego de 30 minutos. (g) Luego de 45 minutos el terreno se cubre en su totalidad teniendo un nivel bastante homogéneo.

5.3 Incendios Forestales

Los incendios forestales consumen una gran cantidad de recursos importantes de la tierra. Se han realizado esfuerzos considerables para la prevención de estos fenómenos, siendo el uso de modelos de incendios forestales uno de las herramientas más importantes. Estos modelos son usados para determinar como se extiende o propaga el fuego bajo las diferentes condiciones del medio ambiente.

Nuevamente, el uso de Sistemas de Información Geográficos (GIS) en modelos de propagación de fuegos forestales han extendido el análisis de áreas quemadas y la determinación de factores de riesgo.

La metodología utilizada para el estudio de fuegos forestales esta basada en reglas que definen el estado de cada parte del terreno. Estas reglas son estocásticas, y están especificadas sobre la base de un modelo semi-empírico. Las simulaciones convencionales de propagación de fuegos forestales están desarrolladas a nivel microscópico, donde se usan ecuaciones dinámicas para determinar la propagación del fuego de un elemento o porción de terreno a otro. El modelo mas conocido en propagación de fuegos forestales es el desarrollado por Rothermel **[Rot72]**. Este permite prever la propagación y la intensidad del fuego sabiendo ciertas propiedades de la vegetación del terreno (matriz de combustible) y condiciones del medio ambiente en las que el fuego ocurrirá.

Tres grupos de parámetros son necesarios para que el modelo pueda determinar la propagación del fuego (RS en ingles Ratio of Spread):

- Propiedades de las partículas o vegetación combustible (contenido de calor, contenido mineral y densidad de las partículas)
- Modelos de combustibles (la vegetación es dividida en clase por tamaño, la profundidad del combustible)
- Valores relacionados al medio ambiente (velocidad del viento, volumen de humedad del combustible e inclinación del terreno).

El modelo de combustible fue previamente tabulado sobre la base de observaciones de determinados terrenos, caracterizando de esta manera el tipo de vegetación posible. Los modelos de combustible utilizados para este trabajo se basan en el denominado modelo NFFL (Nothern Forest Fire Laboratory, recientemente renombrado a Intermountain Fire Sciences Laboratory). Existen 13 clases diferentes de modelos que agrupan diferentes tipos de vegetación. Estos representan la gran mayoría de la situaciones que se pueden encontrar. Cada uno de estos modelos provee todos los valores numéricos requeridos por el modelo de propagación de fuego de Rothermel y se caracterizan de la siguiente manera:

- Modelo 1: Pasto continuo, fino, seco y bajo, con una altura por debajo de la rodilla. La hierba mala y los árboles cubren 1/3 de la superficie. Los fuegos se separan con gran velocidad en este modelo.

- Modelo 2: Pasto continuo, fino, seco y bajo, con la presencia de hierba mala y de los árboles que cubren entre 1/3 y 2/3 de la superficie. Los combustibles son formados por la hierba seca, el follaje y las ramificaciones caídas de la vegetación leñosa. El fuego se propaga rápidamente por el pasto. Las acumulaciones dispersadas del combustible pueden incrementar la intensidad del fuego.

- Modelo 3: Pasto continuo, denso, seco y alto, cerca de 1 metro de altura. 1/3 o más del pasto debe estar seco. Los incendios son mas rápidos y de mayor intensidad.

- Modelo 4: Hierba mala o árboles jóvenes muy densos, con cerca de 2 metros de altura. Continuidad horizontal y vertical del combustible. Abundancia de combustible leñoso muerto (ramas) sobre plantas vivas. El fuego se propaga rápidamente con gran intensidad. La humedad de los combustibles vivos tienen gran influencia en el comportamiento del fuego.

- Modelo 5: Hierba mala densa pero bajas, con una altura no superior a los 0,6 metros. Presenta cargas ligeras de follaje de las misma hierba mala y de pasto, que contribuye para propagar el fuego con los vientos débiles. Fuegos de intensidad moderada.

- Modelo 6: Hierba mala más viejas que la del modelo 5, con alturas entre 0.6 y 1.2 metros. Los combustibles vivos son mas escasos y dispersos. En el conjunto es más inflamable que el del modelo 5. El fuego se propaga a través de la hierba mala con los vientos moderados o fuertes.

- Modelo 7: Hierba mala de una especie muy inflamable, de 0.6 a 2 metros de altura, que propaga al fuego por debajo de los árboles. El fuego se desarrolla sobre el combustible muerto con tasas más altas de humedad que los otros modelos, debido a la naturaleza más inflamable del combustible vivo.

- Modelo 8: Bosque denso de coníferas o follaje (sin hierba mala), con la tierra cubierta por una capa de follaje de dimensiones pequeñas (Ej.: Pino silvestre). Los fuegos son de intensidad débil y avanzan lentamente. Solamente en condiciones meteorológicas desfavorables (altas temperaturas, baja humedad relativa y vientos fuertes), este modelo puede llegar a ser peligroso.

- Modelo 9: Bosque denso de coníferas o de follaje, con la tierra cubierta por una capa poco densa y "ventilada" de follaje de dimensiones más grandes (Ej.: Pinus al pinaster y Castanea sativa). Los fuegos son más rápidos con llamas más grandes que de las del modelo 8.

- Modelo 10: Bosque con gran cantidad de leña y de árboles caídos, como consecuencia de vientos fuertes, de plagas intensas, etc.

- Modelo 11: Bosque poco denso y con algunos herbáceos. Follaje y hierba mala existente ayuda a la propagación del fuego. Los fuegos tienen altas intensidades.

- Modelo 12: Presencia de residuos de exploración más pesados que en el modelo 11, formando una capa continua de una altura más grande (hasta 60 centímetros). Más de la mitad del follaje sigue siendo verdes y atrapadas por las ramificaciones. Los fuegos tendrán altas intensidades.

- Modelo 13: Grandes acumulaciones de residuos de exploración, gruesos, pesados que cubren toda la tierra.

En la Figura 36 se pueden ver ejemplos de los modelos de combustibles.

(a)

(b)

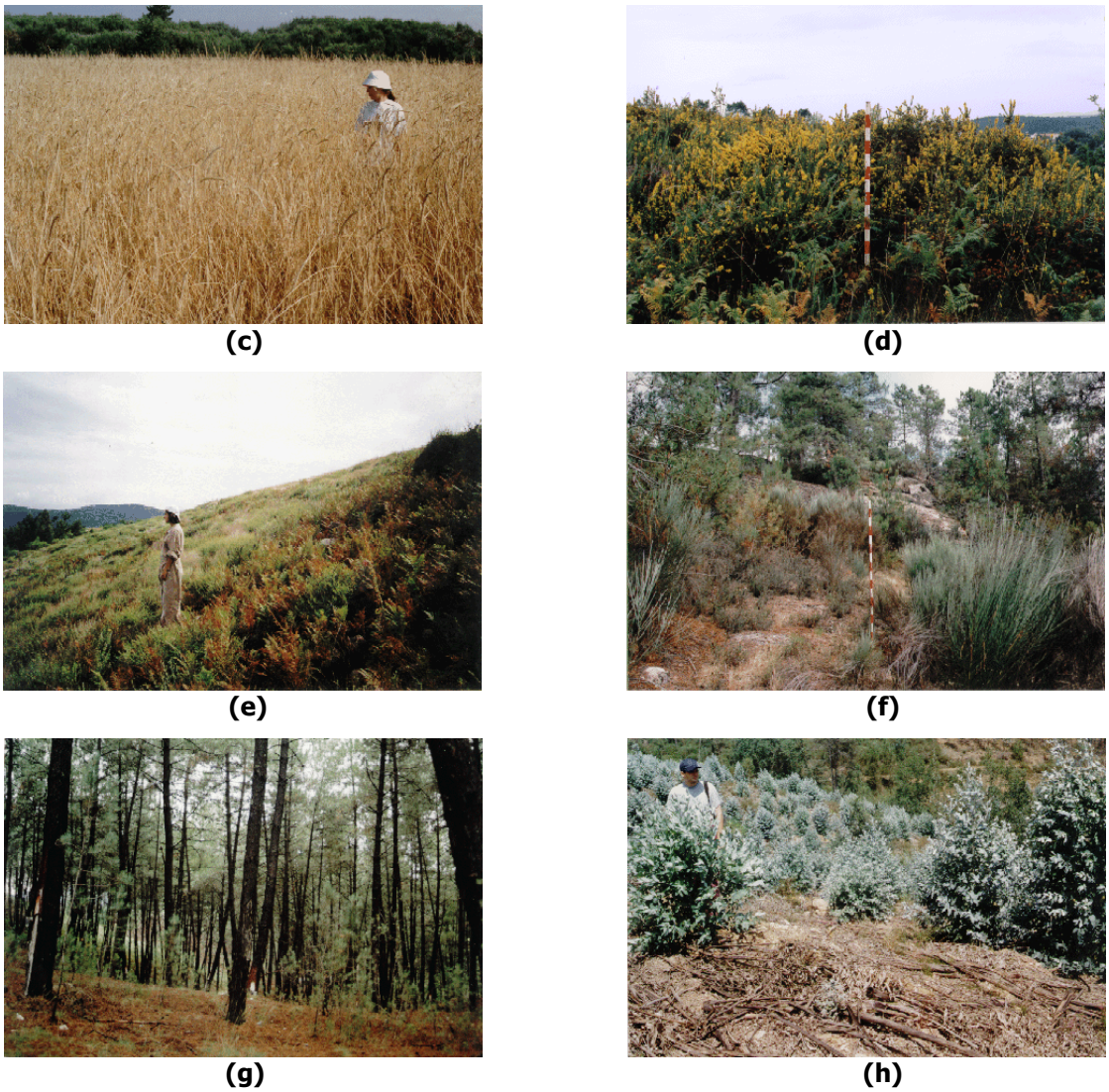


Figura 36 – Fotografías donde se muestran la vegetación en los diferentes modelos de combustibles[1]. (a) Modelo de Combustible 1. (b) Modelo de Combustible 2. (c) Modelo de Combustible 3. (d) Modelo de Combustible 4. (e) Modelo de Combustible 5. (f) Modelo de Combustible 6. (g) Modelo de Combustible 9. (h) Modelo de Combustible 11.

De aplicar el modelo de Rothermel a determinado modelo de combustible y con ciertos valores del medio ambiente se determina la velocidad con que el fuego se propaga, es decir, cuantos metros avanza por minuto.

Debido a que en la versión actual del N-CD++ no se tiene la posibilidad de tener variables asociadas a cada celda de un modelo celular, y la complejidad que implica simular esto agregando varios planos, sumando la variedad de parámetros necesarios, se determino hacer el calculo del modelo de Rothermel de forma externa.

El programa toma como parámetros: el modelo de combustible, la velocidad y dirección del viento, tamaño de cada celda, topología del terreno y dimensión del espacio celular. El programa devuelve un archivo de texto con las velocidades de propagación del fuego en dirección Norte, Noreste, Este, Sudeste,

[1] Información extraída de http://geofogo.cnig.pt/

Sur, Sudoeste, Oeste y Noroeste, para luego ingresar estos valores al N-CD++ y llevar a cabo la simulación del incendio forestal

Para el calculo de los valores se utilizo la librería de funciones FIRELIB[2] con la que se desarrollo el programa. En la Figura 35 se muestra un ejemplo de los resultados obtenidos luego de correr el programa utilizando el modelo de combustible 9, con viento Sudeste y una velocidad de 24.135 kilómetros por hora y cada celda de 15.24x15.24 metros

```
01    Datos y valores obtenidos utilizando el modelo teorico de ROTHERMEL
02    ------------------------------------------------------------
03    Direcci¢n del viento = 115.000000 (angulos en el sentido del reloj con
respecto al norte)
04    Velocidad del viento = 24.135000 [kph]
05    Modelo NFFL = 9
06    Ancho de la celda = 15.240000 [m] (E-O)
07    Alto de la celda  = 15.240000 [m] (N-S)
08    Max. Spread = 13.958213 [mpm]( en dirección donde va el viento)
09
10    0º Spread = 0.221370 [mpm]
11    Distancia = 15.240000 [m]
12
13    45º Spread = 0.469963 [mpm]
14    Distancia = 21.552615 [m]
15
16    90º Spread = 2.744011 [mpm]
17    Distancia = 15.240000 [m]
18
19    135º Spread = 3.844636 [mpm]
20    Distancia = 21.552615 [m]
21
22    180º Spread = 0.533061 [mpm]
23    Distancia = 15.240000 [m]
24
25    225º Spread = 0.234442 [mpm]
26    Distancia = 21.552615 [m]
27
28    270º Spread = 0.165869 [mpm]
29    Distancia = 15.240000 [m]
30
31    315º Spread = 0.163048 [mpm]
32    Distancia = 21.552615 [m]
33
```

Figura 37 – Resultado de la ejecución del Programa que determina la velocidad de avance del fuego.

Una vez obtenidos las tasas de propagación del fuego, en condiciones ambientales y modelos de combustibles predefinido, se desarrollo un modelo celular en N-CD++ que simula el avance del fuego a lo largo del tiempo, determinando así la incidencia en el terreno y los factores de riesgos posibles.

```
01
02 [top]
03 components : ForestFire
04
05 [ForestFire]
06 type : cell
07 dim : (20,20,2)
08 delay : transport
09 defaultDelayTime  : 60000
10 border : nowrapped
11 neighbors : superficie(-1,-1,0) superficie(-1,0,0) superficie(-1,1,0)
```

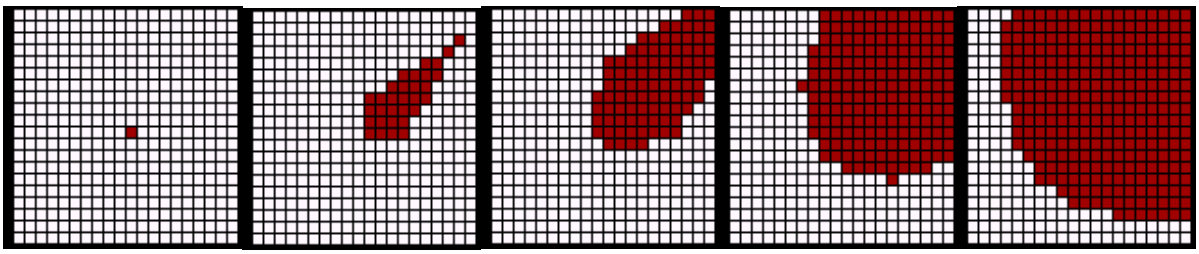
[2] Software de dominio Público derivado directamente del BEHAVE (algoritmo para el análisis del comportamiento de fuego)

```
12 neighbors : superficie(0,-1,0)  superficie(0,0,0)  superficie(0,1,0)
13 neighbors : superficie(1,-1,0)  superficie(1,0,0)  superficie(1,1,0)
14 neighbors :                  superficie(0,0,1)
15 initialvalue : 0
16 initialCellsValue : Fire.val
17 localtransition : FireBehavior
18
19 [FireBehavior]
20 rule : { ((1,-1,0)+(21.552615/17.967136)) } 60000 { cellpos(2)=0 and
   (0,0,0)=0 and (1,-1,0)!=0 and (1,-1,0)!=? and (time/60000)>=((1,-
   1,0)+(21.552615/17.967136))}
21 rule : { ((1,0,0)+(15.24/5.106976)) } 60000 { cellpos(2)=0 and (0,0,0)=0 and
   (1,0,0)!=0 and (1,0,0)!=? and (time/60000)>=((1,0,0)+(15.24/5.106976))}
22 rule : { ((0,-1,0)+(15.24/5.106976)) } 60000 { cellpos(2)=0 and (0,0,0)=0 and
   (0,-1,0)!=0 and (0,-1,0)!=? and (time/60000)>=((0,-1,0)+(15.24/5.106976))}
23 rule : { ((-1,-1,0)+(21.552615/1.872060)) } 60000 { cellpos(2)=0 and
   (0,0,0)=0 and (-1,-1,0)!=0 and (-1,-1,0)!=? and (time/60000)>=((-1,-
   1,0)+(21.552615/1.872060))}
24 rule : { ((1,1,0)+(21.552615/1.872060)) } 60000 { cellpos(2)=0 and (0,0,0)=0
   and (1,1,0)!=0 and (1,1,0)!=? and
   (time/60000)>=((1,1,0)+(21.552615/1.872060))}
25 rule : { ((-1,0,0)+(15.24/1.146091)) } 60000 { cellpos(2)=0 and (0,0,0)=0 and
   (-1,0,0)!=0 and (-1,0,0)!=? and (time/60000)>=((-1,0,0)+(15.24/1.146091))}
26 rule : { ((0,1,0)+(15.24/1.146091)) } 60000 { cellpos(2)=0 and (0,0,0)=0 and
   (0,1,0)!=0 and (0,1,0)!=? and (time/60000)>=((0,1,0)+(15.24/1.146091))}
27 rule : { ((-1,1,0)+(21.552615/0.987474)) } 60000 { cellpos(2)=0 and (0,0,0)=0
   and (-1,1,0)!=0 and (-1,1,0)!=? and (time/60000)>=((-
   1,1,0)+(21.552615/0.987474))}
28
29 rule : { (0,0,0) + 1 } 60000 { cellpos(2)=1 and (0,0,0)<45 }
30 rule : {(0,0,0)} 60000 { t }
31
```

Figura 38 – Especificación del modelo de Fuego forestales

En la Figura 38 se detalla la especificación del modelo en N-CD++. Se utilizaron los valores (Figura 37) que resultaron del calculo. El modelo contiene 2 planos, en el primer plano o plano 0 residen las celdas que representan el terreno con vegetación, en el plano 1 las celdas que son utilizadas para determinar el paso del tiempo. Luego en la línea 19 comienza la descripción de las reglas del comportamiento, cada una de ellas analiza lo siguiente: corrobora que la celda en análisis no tenga la presencia de fuego, luego analiza el estado de un vecino determinado, si este tiene la presencia de fuego entonces se calcula cuanto tiempo llevara al fuego a avanzar hasta la celda en análisis, como el valor de la celda vecina representa el momento del tiempo donde comenzó el incendio en esa celda, se puede determinar tomando el tiempo actual y la velocidad del avance del fuego, en que momento del tiempo el fuego alcanzo a la celda en análisis, y ese valor es el nuevo valor que toma la celda.

Debido a que la velocidad del avance del fuego es diferente dado que las condiciones del medio (vientos fuertes, etc.) pueden favorecer el avance del fuego en determinada dirección, por eso se tienen 8 reglas, una por cada vecino cercano. En la Figura 39 se pueden visualizar los resultados del modelo.



(a)

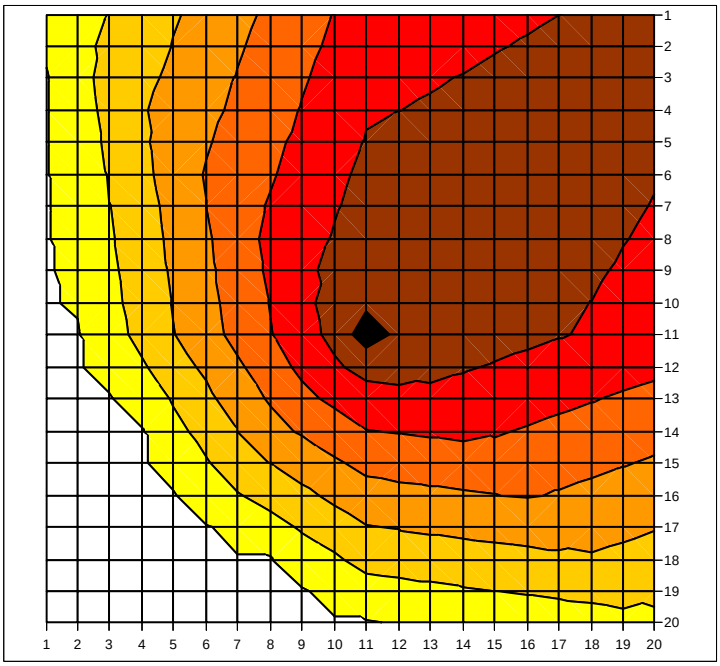


Figura 39 – (a) Avance del fuego utilizando GrafLog. El fuego se origina en la celda (11,11) con viento en dirección Noreste. (b) Gráfico que representa el avance del fuego a lo largo de 2 horas. El fuego se origina en la zona de color más oscuro. Cada cambio de color representa 20 minutos

5.4 Solidificación Binaria

En esta sección se presenta un modelo celular bidimensional, el cual simula el proceso de solidificación binaria **[Kre97]**. La simulación consiste en la solidificación de partículas de Cloruro de Amonio (NH_4Cl) en una solución acuosa (H_2O), observando las estructuras de cristales, la velocidad a la que se generan y los patrones de las mismas.

El modelo analiza el estado (sólido o liquido) y la cantidad de soluto (volumen% H_2O) en cada partícula o elemento. Los valores de cada celda oscilan entre los valores reales 0.0 y 1.0, correspondiendo a la cantidad de soluto que contenga cada una de ellas. Las celdas sólidas tendrán una concentración de soluto igual a 0.0. El algoritmo que describe el proceso de solidificación es el siguiente:

a. en primer paso identificar todas las partículas que estén en estado liquido
b. una vez evaluada cada partícula, basándose en la concentración de soluto de la misma se analiza la posibilidad de solidificar o no.
c. Cada partícula que solidifique debe expulsar toda la cantidad de soluto que posee a sus vecinos líquidos, debido a que en la partículas sólidas el soluto no se puede depositar. En el caso de que no existan vecinos líquidos, la partícula no solidificara, debido a que se tiene que mantener el equilibrio del sistema (manteniendo las cantidades de H_2O)
d. Una vez analizada toda la superficie, se debe someter la misma a un modelo de difusión de soluto sobre las partículas liquidas, para equilibrar el contenido de soluto en el sistema.

Claramente el punto (a) es muy fácil de implementar, dado que solamente hay que elegir aquellas celdas cuyos valores sean distintos de 1.0. En el punto (b) es necesario conocer cuando una celda en estado liquido pasa a estar en estado sólido. Para esto es necesario determinar aspectos físicos (surface tension, interface curvature y crystaline anisotropy) que juegan un rol importante en el proceso. Primero

debe ser calculado el equilibrio de la concentración, para este proceso se debe tener en cuenta los primeros y segundos vecinos cercanos, es decir, los vecinos que se encuentran a un radio de 2 celdas de distancia. Los vecinos que se tendrán en cuenta son solo los que están en estado sólido. Formalmente se define como:

$$C_{eq} = C_l - \gamma(flat - \sum_{\substack{k \in vecinos(1) \\ k \in vecinos(2)}} (\phi_k \varepsilon_k))$$

donde:

C_{eq} = el equilibrio de la concentración
C_l = el equilibrio de la concentración en una interface plana (flat)
γ = se denomina surface tension
ϕ_k = es el estado sólido o liquido del vecino k
ε_k = es la distancia del vecino k en celdas, para los vecinos mas cercanos es 1
flat = es la mitad de la suma de todas las distancias entre todos los vecinos, en este caso tomara el valor de 20 debido a que tenemos 8 vecinos a distancia 1 y 16 vecinos a distancia 2, esto suma 40.

Una vez calculado el equilibro de la concentración, si se determina que la celda en análisis esta bajo equilibrio de soluto (H_2O), la celda solidificara expulsando el contenido de soluto a los vecinos.

El paso siguiente, el paso d, es donde se deberá difundir todo el soluto expulsado por las celdas que están en condiciones de solidificar, siempre y cuando exista una cantidad de vecinos en estado liquido que absorban la cantidad expulsada, de otro modo la celda no podrá congelar. Para esto se utiliza la siguiente ecuación:

$$C_{nuevo} = C_{viejo}(1 - 8\lambda) + \lambda \sum_{k \in vecinos(1)} \phi_k$$

donde:

C_{nuevo} = representa la cantidad de soluto que tendrá la celda luego de recibir la cantidad expulsada por los vecinos que solidificaron
C_{viejo} = cantidad de soluto actual
λ = constante de difusividad (debe ser menor a 0.25 y depende de la distancia entre cada celda y el tiempo de demora entre cada ejecución del algoritmo de difusión
ϕ_k = es el estado sólido (= 1) o liquido (= 0) del vecino k

Este ultimo proceso se deberá realizar una cantidad de veces para lograr una difusión mas uniforme.

```
01 [top]
02 components : campo
03
04 [campo]
05 type : cell
06 dim : (30,30,2)
07 delay : transport
08 defaultDelayTime  : 100
09 border : nowrapped
10 neighbors : campo(-2,-2,0) campo(-2,-1,0) campo(-2,0,0) campo(-2,1,0) campo(-
   2,2,0)
11 neighbors : campo(-1,-2,0) campo(-1,-1,0) campo(-1,0,0) campo(-1,1,0) campo(-
   1,2,0)
12 neighbors : campo(0,-2,0) campo(0,-1,0) campo(0,0,0) campo(0,1,0)
   campo(0,2,0)
13 neighbors : campo(1,-2,0) campo(1,-1,0) campo(1,0,0) campo(1,1,0)
   campo(1,2,0)
14 neighbors : campo(2,-2,0) campo(2,-1,0) campo(2,0,0) campo(2,1,0)
   campo(2,2,0)
```

```
15 neighbors :                                campo(0,0,1)
16 initialvalue : 0.65
17 initialCellsValue : celdas30.val
18 localtransition : calculus
19
20 [calculus]
21 rule : { IF( fractional((0,0,0)) < ( 0.8  - ( 0.01 * ( 20 - ((2 *
   if(isUndefined((-2,-2,0)),0,trunc((-2,-2,0)))) + (2 * if(isUndefined((-2,-
   1,0)),0,trunc((-2,-1,0)))) + (2 * if(isUndefined((-2,0,0)),0,trunc((-
   2,0,0)))) + (2 * if(isUndefined((-2,1,0)),0,trunc((-2,1,0)))) + (2 *
   if(isUndefined((-2,2,0)),0,trunc((-2,2,0)))) + (2 * if(isUndefined((-1,-
   2,0)),0,trunc((-1,-2,0)))) + (2 * if(isUndefined((-1,2,0)),0,trunc((-
   1,2,0)))) + (2 * if(isUndefined((0,-2,0)),0,trunc((0,-2,0)))) + (2 *
   if(isUndefined((0,2,0)),0,trunc((0,2,0)))) + (2 * if(isUndefined((1,-
   2,0)),0,trunc((1,-2,0)))) + (2 * if(isUndefined((1,2,0)),0,trunc((1,2,0)))) +
   (2 * if(isUndefined((2,-2,0)),0,trunc((2,-2,0)))) + (2 * if(isUndefined((2,-
   1,0)),0,trunc((2,-1,0)))) + (2 * if(isUndefined((2,0,0)),0,trunc((2,0,0)))) +
   (2 * if(isUndefined((2,1,0)),0,trunc((2,1,0)))) + (2 *
   if(isUndefined((2,2,0)),0,trunc((2,2,0)))) + (1 * if(isUndefined((-1,-
   1,0)),0,trunc((-1,-1,0)))) + (1 * if(isUndefined((-1,0,0)),0,trunc((-
   1,0,0)))) + (1 * if(isUndefined((-1,1,0)),0,trunc((-1,1,0)))) + (1 *
   if(isUndefined((0,-1,0)),0,trunc((0,-1,0)))) + (1 *
   if(isUndefined((0,1,0)),0,trunc((0,1,0)))) + (1 * if(isUndefined((1,-
   1,0)),0,trunc((1,-1,0)))) + (1 * if(isUndefined((1,0,0)),0,trunc((1,0,0)))) +
   (1 * if(isUndefined((1,1,0)),0,trunc((1,1,0)))))))))), (2 +
   fractional((0,0,0))) , (0,0,0) )} 100 { Cellpos(2) = 0 and (0,0,1) = 1 and
   trunc((0,0,0)) = 0}
22
23 rule : { IF(((if(IsUndefined((-1,-1,0)) or trunc((-1,-1,0)) = 0,0,1)) +
   (if(IsUndefined((-1,0,0)) or trunc((-1,0,0)) = 0,0,1)) + (if(IsUndefined((-
   1,1,0)) or trunc((-1,1,0)) = 0,0,1)) + (if(IsUndefined((0,-1,0)) or
   trunc((0,-1,0)) = 0,0,1)) + (if(IsUndefined((0,1,0)) or trunc((0,1,0)) =
   0,0,1)) + (if(IsUndefined((1,-1,0)) or trunc((1,-1,0)) = 0,0,1)) +
   (if(IsUndefined((1,0,0)) or trunc((1,0,0)) = 0,0,1)) +
   (if(IsUndefined((1,1,0)) or trunc((1,1,0)) = 0,0,1))) = 8,
   fractional((0,0,0)) ,(0,0,0))} 100 { cellpos(2) = 0 and (0,0,1) = 2 and
   trunc((0,0,0)) = 2}
24
25 rule : { (if(IsUndefined((-1,-1,0)) or trunc((-1,-1,0)) = 0,0,1)) +
   (if(IsUndefined((-1,0,0)) or trunc((-1,0,0)) = 0,0,1)) + (if(IsUndefined((-
   1,1,0)) or trunc((-1,1,0)) = 0,0,1)) + (if(IsUndefined((0,-1,0)) or
   trunc((0,-1,0)) = 0,0,1)) + (if(IsUndefined((0,1,0)) or trunc((0,1,0)) =
   0,0,1)) + (if(IsUndefined((1,-1,0)) or trunc((1,-1,0)) = 0,0,1)) +
   (if(IsUndefined((1,0,0)) or trunc((1,0,0)) = 0,0,1)) +
   (if(IsUndefined((1,1,0)) or trunc((1,1,0)) = 0,0,1)) + (0,0,0) } 100 {
   cellpos(2) = 0 and (0,0,1) = 3 and trunc((0,0,0)) = 2}
```

```
26 rule : { if(((if(IsUndefined((-1,-1,0)) or trunc((-1,-1,0)) <
   2,0,fractional((-1,-1,0))/(trunc((-1,-1,0)) - 2))) + (if(IsUndefined((-
   1,0,0)) or trunc((-1,0,0)) < 2,0,fractional((-1,0,0))/(trunc((-1,0,0)) - 2)))
   + (if(IsUndefined((-1,1,0)) or trunc((-1,1,0)) < 2,0,fractional((-
   1,1,0))/(trunc((-1,1,0)) - 2))) + (if(IsUndefined((0,-1,0)) or trunc((0,-
   1,0)) < 2,0,fractional((0,-1,0))/(trunc((0,-1,0)) - 2))) +
   (if(IsUndefined((0,1,0)) or trunc((0,1,0)) <
   2,0,fractional((0,1,0))/(trunc((0,1,0)) - 2))) + (if(IsUndefined((1,-1,0)) or
   trunc((1,-1,0)) < 2,0,fractional((1,-1,0))/(trunc((1,-1,0)) - 2))) +
   (if(IsUndefined((1,0,0)) or trunc((1,0,0)) <
   2,0,fractional((1,0,0))/(trunc((1,0,0)) - 2))) + (if(IsUndefined((1,1,0)) or
   trunc((1,1,0)) < 2,0,fractional((1,1,0))/(trunc((1,1,0)) - 2))) +
   fractional((0,0,0))) > 1,(99+fractional((0,0,0))),((if(IsUndefined((-1,-1,0))
   or trunc((-1,-1,0)) < 2,0,fractional((-1,-1,0))/(trunc((-1,-1,0)) - 2))) +
   (if(IsUndefined((-1,0,0)) or trunc((-1,0,0)) < 2,0,fractional((-
   1,0,0))/(trunc((-1,0,0)) - 2))) + (if(IsUndefined((-1,1,0)) or trunc((-
   1,1,0)) < 2,0,fractional((-1,1,0))/(trunc((-1,1,0)) - 2))) +
   (if(IsUndefined((0,-1,0)) or trunc((0,-1,0)) < 2,0,fractional((0,-
   1,0))/(trunc((0,-1,0)) - 2))) + (if(IsUndefined((0,1,0)) or trunc((0,1,0)) <
   2,0,fractional((0,1,0))/(trunc((0,1,0)) - 2))) + (if(IsUndefined((1,-1,0)) or
   trunc((1,-1,0)) < 2,0,fractional((1,-1,0))/(trunc((1,-1,0)) - 2))) +
   (if(IsUndefined((1,0,0)) or trunc((1,0,0)) <
   2,0,fractional((1,0,0))/(trunc((1,0,0)) - 2))) + (if(IsUndefined((1,1,0)) or
   trunc((1,1,0)) < 2,0,fractional((1,1,0))/(trunc((1,1,0)) - 2))) + (0,0,0)))}
   100 { cellpos(2) = 0 and (0,0,1) = 4 and trunc((0,0,0)) = 0}
27
28 rule : { if((if(trunc((-1,-1,0)) = 99,1,0) + if(trunc((-1,0,0)) = 99,1,0) +
   if(trunc((-1,1,0)) = 99,1,0) + if(trunc((0,-1,0)) = 99,1,0) +
   if(trunc((0,1,0)) = 99,1,0) + if(trunc((1,-1,0)) = 99,1,0) +
   if(trunc((1,0,0)) = 99,1,0) + if(trunc((1,1,0)) = 99,1,0)) > 1,
   fractional((0,0,0)),(0,0,0))} 100 { cellpos(2) = 0 and (0,0,1) = 5 and
   trunc((0,0,0)) = 2}
29
30 rule : 1 100 { cellpos(2) = 0 and (0,0,1) = 6 and trunc((0,0,0)) > 1 and
   trunc((0,0,0)) != 99 }
31 rule : { fractional((0,0,0)) } 100 {cellpos(2) = 0 and (0,0,1) = 7 and
   trunc((0,0,0)) = 99 }
32
33 rule : { ((1 - (0.00001 * (if(IsUndefined((-1,-1,0)) or trunc((-1,-1,0)) =
   1,0,1) + if(IsUndefined((-1,1,0)) or trunc((-1,1,0)) = 1,0,1) +
   if(IsUndefined((1,1,0)) or trunc((1,1,0)) = 1,0,1) + if(IsUndefined((1,-1,0))
   or trunc((1,-1,0)) = 1,0,1))) - (0.00001 * (if(IsUndefined((-1,0,0)) or
   trunc((-1,0,0)) = 1,0,1) + if(IsUndefined((0,1,0)) or trunc((0,1,0)) = 1,0,1)
   + if(IsUndefined((1,0,0)) or trunc((1,0,0)) = 1,0,1) + if(IsUndefined((0,-
   1,0)) or trunc((0,-1,0)) = 1,0,1)))) * fractional((0,0,0))) + (0.00001 *
   (if(IsUndefined((-1,-1,0)) or trunc((-1,-1,0))= 1,0,fractional((-1,-1,0))) +
   if(IsUndefined((-1,1,0)) or trunc((-1,1,0)) = 1,0,fractional((-1,1,0))) +
   if(IsUndefined((1,1,0)) or trunc((1,1,0)) = 1,0,fractional((1,1,0))) +
   if(IsUndefined((1,-1,0)) or trunc((1,-1,0)) = 1,0,fractional((1,-1,0))))) +
   (0.00001 * (if(IsUndefined((-1,0,0)) or trunc((-1,0,0)) = 1,0,fractional((-
   1,0,0))) + if(IsUndefined((0,1,0)) or trunc((0,1,0)) =
   1,0,fractional((0,1,0))) + if(IsUndefined((1,0,0)) or trunc((1,0,0)) =
   1,0,fractional((1,0,0))) + if(IsUndefined((0,-1,0)) or trunc((0,-1,0)) =
   1,0,fractional((0,-1,0))))))} 100 { cellpos(2) = 0 and (0,0,1) >= 8 and
   trunc((0,0,0)) = 0 }
34
35 rule : { (0,0,0) + 1 } 100 { (0,0,0) <= 15 and (0,0,0) >= 1 and Cellpos(2) =
   1 }
36 rule : 1 100 { Cellpos(2) = 1 }
37 rule : { (0,0,0) } 100 { t }
38
```

Figura 40 – Especificación del modelo de Solidificación Binaria

En la Figura 40 se puede observar la especificación del modelo. El modelo consiste solamente de un único componente, un modelo celular bidimensional de 30x30 celdas con 2 planos. En el plano 0 se tienen

los valores del estado (sólido o liquido) de cada partícula del modelo, y el plano 1 se utiliza para saber en que etapa se esta del proceso. La descripción de las reglas que rige el comportamiento del modelo comienza en la línea 20. Se puede observar que las celdas son extensas, esto se debe a que la actual versión del N-CD++ solo soporta valores reales, y se necesitan llevar mas de un valor por cada celda a lo largo de la simulación. Para lograr esto se utiliza la parte entera del valor real para guardar el estado (sólido o liquido) y la parte decimal, se utiliza para guardar el porcentaje de soluto que posee esa celda. Para poder determinar la parte entera del valor de una celda se utiliza la función **trunc** y para determinar la parte decimal se utiliza la función **fractional**. Por esto se extienden las reglas dado que hay que preguntar en todo los casos que se necesiten saber si una celda esta o no en estado sólido o liquido por su parte entera, y no solo hace que sean extensas las reglas sino también que baja la performance de la herramienta.

La regla que se encuentra en la línea 21 será evaluada si el valor de la celda del plano 1 es igual a 1. Esta regla sirve para calcular el equilibrio de la concentración (C_{eq}), los valores sugeridos utilizados por los parámetros son los siguientes: C_l = 0.8 (es decir 80% de soluto) y γ = 0.0095. Una vez que se determina que la celda esta en equilibrio se la marca colocando un 2 en la parte entera del valor de la celda, dado que se deben analizar otros aspectos antes de congelar la misma. La siguiente regla, línea 23, sirve para analizar si las celdas que están marcadas como con posibilidades de congelar tiene algún vecino en estado liquido, dado que si esto no ocurre la celda no podrá congelar debido a que no tiene donde colocar la cantidad de soluto que expulsa.

En la línea 25, se tiene el siguiente paso del algoritmo. Se debe saber por cada celda en condiciones de solidificar la cantidad de celdas que están en estado liquido, esto sirve para que cuando se expulse la cantidad de soluto que posee la celda sea de forma isotrópica, es decir, se le asigne a cada celda vecina en estado liquido una cantidad igual y proporcional.

La línea 26 posee la regla que es la encarga de verificar que la cantidad de soluto que recibe cada celda sea una cantidad no mayor al 100% del volumen, dado que físicamente no se puede ocupar mas del 100% del volumen de un objeto. Llegado el caso que una celda reciba mas del 100% de soluto las celdas vecinas que estaban en condición de solidificar deben abortar esa condición dado que no tienen donde depositar la cantidad de soluto que deben expulsar para solidificar. Toda celda que viole el principio físico se le coloca en el valor entero de la misma el numero 99, que sirve como una marca.

En la línea 28 se pasa a todas las celdas que estaban en condición de congelar, y tengan un vecino marcado con 99 en la parte entera del valor de la celda, a estado liquido.

La regla de la línea 30 pasa a estado sólido a aquellas celdas que pasaron todas las condiciones anteriores. En la regla siguiente se borran todas las marcar que existan, para luego, en la línea 33 generar el proceso de difusión del soluto a través de toda la superficie. Esta operación se repite 15 veces como lo indica la regla de la línea 35. Por ultimo las reglas que siguen se utilizan para ir actualizando el contador de pasos que se encuentra en el plano 1 y sirve de referencia para saber que reglas se deben analizar.

El la figura 41 se pueden observar los resultados de la simulación. Se comienza congelando las 4 celdas centrales.

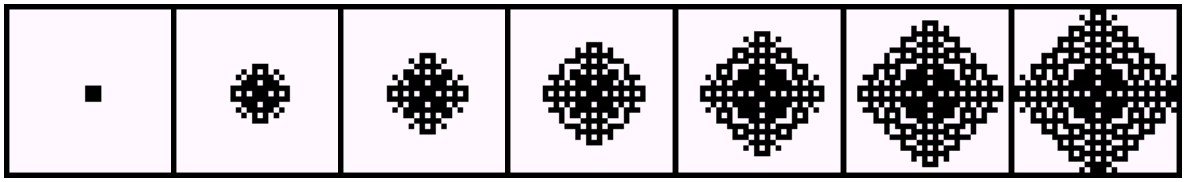


Figura 41 – Estructura de los cristales

6. Conclusiones

En este trabajo se presentaron dos extensiones a la herramienta N-CD++ utilizadas para el modelado y simulación de modelos Cell-DEVS. La primera extensión permite que se puedan trabajar modelos con diferentes geometrías tales como triangular y hexagonal. La otra permite visualizar los resultados de las simulaciones de manera gráfica. Las extensiones desarrolladas a la herramienta permiten representar nuevos modelos los cuales, debido a las limitaciones del N-CD++, eran imposibles de simular.

Luego se definieron modelos sencillos utilizando dichas extensiones que demostraron los nuevos alcances de la herramienta.

Por ultimo se desarrollaron modelos complejos donde se puso de manifiesto el universo de modelos posibles que pueden ser tratados con el conjunto de herramientas desarrollado.

Cabe considerar que como las herramientas y programas adicionales fueron construidos usando una versión estándar de C++, favoreció la portabilidad de la misma a distintas plataformas sin costo adicional. Hasta el momento N-CD++ fue testeado en forma exitosa en Windows 95 / 98 / NT, Linux, AIX, HP-UX y Sun, lo que permite a los usuarios a realizar simulaciones en su ámbito habitual para luego poder realizar estudios más profundos con máquinas con mayor poder de cómputo.

Las herramientas y modelos son de dominio publico y pueden ser accedidas en: "http://www.dc.uba.ar/people/proyinv/celldevs".

7. Bibliografía

[BBW98] BARYLKO, A.; BEYOGLONIAN, J.; WAINER, G. "GAD: a General Application DEVS environment". *Proceedings of IASTED Applied Modeling and Simulation 1998.* Hawaii, U.S.A. 1998.

[GM76] GIAMBIASI, N.; MIARA, A. "SILOG: A practical tool for digital logic circuit simulation. Proceedings of the 16th. D.A.C., San Diego, U.S.A. 1976.

[Gia96] GIAMBIASI, N. "Introduction a la modélisation et a la simulation". Materiales del curso D.E.A.; , Université d'Aix-Marseille III. 1996.

[Kre97] KREMEYER, K. "Experimental and Computational Investigations of Binary Solidification". A dissertation submitted to the Faculty of the Departament of Physics. The University of Arizona. 1997

[Moo96] MOON, Y.; ZEIGLE, B.; BALL, G.; GUERTIN, D., 1996. *"DEVS representation of spatially distributed systems: validity, complexity reduction"*. IEEE Transactions on Systems, Man and Cybernetics. pp. 288-296.

[Rot72] Rothermel, Richard C. "A mathematical model for predicting fire spread in wildland fuels". Research Paper INT-115. Ogden,UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station; 1972. 40 p.

[RW99] RODRIGUEZ, D.; WAINER, G. "New Extensions to the CD++ tool". In *Proceedings of SCS Summer Multiconference on Computer Simulation*. 1999.

[Tof87] TOFFOLI T.; MARGOLUS N. "Cell space Machines, A new environment for modeling". The MIT Press. Cambridge, Massachusetts. London, England. 1987

[Wai98] WAINER, G. "Discrete-events cellular models with explicit delays". Ph.D. Thesis, Université d'Aix-Marseille III. 1998.

[Weim97] WEIMAR, J. "*Simulation with Cell spaces*". Berlin, Logos – Verl, 1997

[WFG97] WAINER,G.; FRYDMAN, C.; GIAMBIASI, N. "An environment to simulate cellular DEVS models". *Proceedings of the SCS European Multiconference on Simulation*. Istanbul, Turkey. 1997.

[WFG97b] WAINER,G.; GIAMBIASI, N.; FRYDMAN, C. "Cell-DEVS models with transport and inertial delays". *Proceedings of the 9th. European Simulation Symposium and Exhibition*. Passau, Germany. 1997.

[Zei76] ZEIGLER, B. "Theory of modeling and simulation". Wiley, 1976.

[Zei84] ZEIGLER, B. "Multifaceted Modeling and discrete event simulation". Academic Press, 1984.

[Zei90] ZEIGLER, B. "Object-oriented simulation with hierarchical modular models". Academic Press, 1990.

[Zei95] ZEIGLER, B. "Object-oriented simulation with hierarchical modular models". Revised to include source code for DEVS-C++. Technical Report. Department of Electrical and Computer Engineering. University of Arizona. 1995.
[Zei95b] ZEIGLER, B.; KIM, D. "Design of high level modeling / high performance simulation environments". Technical Report, Department of Electrical and Computer Engineering, University of Arizona. 1995.
[Zei96] ZEIGLER, B.; MOON, Y.; KIM, D.; KIM, D. "DEVS-C++: A high performance modeling and simulation environment". Technical Report, Department of Electrical and Computer Engineering, University of Arizona. In Proceedings of 29th. Hawaii International Conference on System Sciences, Jan. 1996.

8. Bibliografía Adicional

[Gar70] GARDNER, M. "*The Fantastic Combinations of John Conway's New Solitaire Game 'Life'*". Scientific American, 23 (4), 1970, pp. 120-123.
[WG00] WAINER, G.; GIAMBIASI, N. "Timed Cell-DEVS: modeling and simulation of cell spaces ". Invited paper for the book "Discrete Event Modeling & Simulation: Enabling Future Technologies", to be published by Springer-Verlag. 2000.
[Wol86] WOLFRAM, S. "Theory and applications of cell space". Vol.1, Advances Series on Complex Systems. World Scientific, Singapore, 1986.
[ZTP00] ZEIGLER, B.; KIM, T.; PRAEHOFER, H. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". Academic Press. 2000.

Apéndice

Lenguaje de Especificación de N–CD++

La sintaxis del lenguaje usado por *N-CD++* para la especificación del comportamiento de los modelos celulares atómicos puede definirse con la siguiente BNF, donde las palabras escritas con letras minúsculas y en negrita representan terminales, mientras que las escritas en mayúsculas representan no terminales:

```
RULELIST             = RULE
                     |  RULE RULELIST

RULE           = RESULT RESULT { BOOLEXP }

RESULT         = CONSTANT
                  |  { REALEXP }

BOOLEXP        = BOOL
                  |  ( BOOLEXP )
                  |  REALRELEXP
                  |  not BOOLEXP
                  |  BOOLEXP OP_BOOL BOOLEXP

OP_BOOL        = and  |  or  |  xor  |  imp  |  eqv

REALRELEXP     = REALEXP OP_REL REALEXP
                  |  COND_REAL_FUNC(REALEXP)

REALEXP        = IDREF
                  |  ( REALEXP )
                  |  REALEXP OPER REALEXP

IDREF          = CELLREF
                  |  CONSTANT
                  |  FUNCTION
```

```
                  |  portValue(PORTNAME)
                  |  send(PORTNAME, REALEXP)
                  |  cellPos(REALEXP)

CONSTANT             = INT
                  |  REAL
                  |  CONSTFUNC
                  |  ?

FUNCTION             = UNARY_FUNC(REALEXP)
                  |  WITHOUT_PARAM_FUNC
                  |  BINARY_FUNC(REALEXP, REALEXP)
                  |  if(BOOLEXP, REALEXP, REALEXP)
                  |  ifu(BOOLEXP, REALEXP, REALEXP, REALEXP)

CELLREF       = (INT, INT RESTO_TUPLA

RESTO_TUPLA   = , INT RESTO_TUPLA
                |  )

BOOL          = t  |  f  |  ?

OP_REL        = !=  |  =  |  >  |  <  |  >=  |  <=

OPER          = +  |  -  |  *  |  /

INT           = [SIGN] DIGIT {DIGIT}

REAL          = INT  |  [SIGN] {DIGIT}.DIGIT {DIGIT}

SIGN          = +  |  -

DIGIT         = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

PORTNAME            = thisPort  |  STRING

STRING        = LETTER {LETTER}

LETTER        = a | b | c |...| z | A | B | C |...| Z

CONSTFUNC     = pi | e | inf | grav | accel | light | planck | avogadro |
                faraday | rydberg | euler_gamma | bohr_radius | boltzmann |
                bohr_magneton | golden | catalan | amu | electron_charge |
                 ideal_gas | stefan_boltzmann | proton_mass | electron_mass |
                neutron_mass  | pem

WITHOUT_PARAM_FUNC = truecount | falsecount | undefcount | time | random |
                     randomSign

UNARY_FUNC    = abs | acos | acosh | asin | asinh | atan | atanh | cos |
                sec | sech | exp | cosh | fact | fractional | ln | log |
                round | cotan | cosec | cosech | sign | sin | sinh |
                statecount | sqrt | tan | tanh | trunc | truncUpper |
                poisson | exponential | randInt | chi | asec | acotan |
                asech | acosech | nextPrime | radToDeg | degToRad |
                nth_prime | acotanh | CtoF | CtoK | KtoC | KtoF | FtoC |
                FtoK

BINARY_FUNC   = comb | logn | max | min | power | remainder | root | beta |
                gamma | lcm | gcd | normal | f | uniform | binomial |
                rectToPolar_r | rectToPolar_angle | polarToRect_x | hip |
                polarToRect_y

COND_REAL_FUNC = even | odd | isInt | isPrime | isUndefined
```

Manual del usuario del GRAFLOG

El primer paso es obtener el archivo de salida con los resultados que será utilizado por el graflog. Si se supone que se cuenta con el siguiente modelo model.ma, la llamada al simulador seria la siguiente

Simu –mmodel.ma –lsalida.log ...

Esto dejaría como resultado un archivo log que es utilizado por otra herramienta del set N-CD++, el drawlog. El mismo se encarga de interpretar este archivo y convertirlo a una grilla de valores numéricos.

El drawlog posee un switch –f# que permite obtener una salida sin formatos, es decir, solo los valores de cada celda y donde # es el plano que se desea obtener en un modelo tridimensional, dado que en modo texto, solo planos pueden ser visualizados por pantalla. Vale aclarar que el Graflog solo graficará modelos bidimensionales.

Siguiendo el ejemplo anterior, se debe llamar al drawlog de la siguiente forma:

Drawlog –mmodel.ma –lsalida.log ..... –f1 > model.drw

Como se puede apreciar, se redirecciona la salida del drawlog a un archivo (model.drw).Una vez que ya se tiene este archivo, se puede ver su representación de forma gráfica.

Graflog posee los siguientes switchs:

-x# donde # es un entero que representa el alto de la grilla que figura en el modelo

-y# donde # es un entero que representa el ancho de la grilla en el modelo

-e# es la escala. La escala 1 = 1x1pixel ; 2 = 2x2pixeles ; etc.

-fxxxxx donde xxxxx es el nombre del archivo que deja como salida el drawlog en el ejemplo anterior seria -fmodel.drw

-exxxxx donde xxxxx es el nombre de archivo que contiene la tabla de colores

-s# donde # representa al número de avances en el tiempo, es decir, si se supone que el modelo posee una actualización de la grilla completa en t unidades de tiempo y se corre la simulación durante 10*t unidades de tiempos. Por otro lado, si se corre el Graflog –s2, hará que el gráfico se actualizara hasta 2*t unidades de tiempo y pare, esperando la intervención del usuario para que presione una tecla, luego de presionada la misma el graficador seguirá graficando hasta 4*t y nuevamente parará. Así hasta alcanzar 10*t. Con esto se puede concluir que si se coloca –s1 la simulación sé graficará paso a paso. Si no se coloca el mismo se graficará continuamente hasta alcanzar el tiempo final.

-t# Similar a –s pero solamente avanza # unidades de tiempo y luego avanza paso a paso.

-v Permite vizualizar la grilla..

Si se desea que el graficador avance sin intervención del usuario se deberá presionar la tecla ESC, esto hará que el tiempo avance sin que el programa le pida al usuario que presione una tecla.

El archivo que contiene la tabla de colores es un archivo de texto con el siguiente formato:

X1 Y1
X2 Y2
X3 Y3

... ...
X30 Y30

donde X es un Rango que representara los valores posibles de la celda a graficar e Y será el color asignado al mismo en el gráfico. El rango tiene el siguiente formato

[###;###] ó (###;###) ó [###;###) ó (###;###]
donde ### es cualquier número real (float) incluyendo a +INF y –INF

Ejemplos de Rangos:
(-INF;20)
[-23;+INF]
[0;0]
[45;47.8994004)

Están permitidas hasta 30 asignaciones de colores. La lista de colores permitidos es la siguiente:

black	0
blue	1
green	2
cyan	3
red	4
magenta	5
brown	6
lightgray	7
darkgray	8
lightblue	9
lightgreen	10
lightcyan	11
lightred	12
lightmagenta	13
yellow	14
white	15

Datos importantes para tener en cuenta son:

1 – Los valores por defecto son los siguientes:
- x = 100
- y = 100
- e = 1
- s = sin interrupciones
- f = "salidaF.drw"
- c = "color.txt"

2 – Se debe tener siempre una asignación de colores dado que sino no se verán resultados