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

Tesis de Licenciatura

Definición de extensiones a un lenguaje de microsimulación para tráfico urbano

Tutor:

Dr. Gabriel A. Wainer

Autores:

Andrea R. Díaz
(adiaz@dc.uba.ar**)**

Verónica I. Vazquez
(vvazquez@dc.uba.ar**)**

2000

Índice

Sección I: Introducción y objetivos

El crecimiento del tráfico urbano ha traído serios y preocupantes problemas de congestión en muchas ciudades del mundo. Como, frecuentemente, la demanda se incrementa en una proporción mayor al agregado de capacidad, la situación continuará empeorando, a menos que, sean implementadas mejores estrategias de manejo de tráfico.

Las decisiones tomadas para enfrentar este problema requieren que se posea información confiable sobre la situación actual y deben basarse en un cuidadoso análisis.

En los últimos años, la simulación se ha convertido en una herramienta comúnmente usada para entender las características de un sistema de tráfico y permitir, en consecuencia, elegir un diseño apropiado que combine todos los elementos que influyen en su comportamiento (sentido de circulación de los vehículos, señales, semáforos, normas de tránsito, sistemas de información inteligentes, etc), de manera de lograr la optimización del flujo de vehículos.

Existen diversas herramientas y técnicas de simulación aplicables a este problema. El formalismo de Autómatas Celulares es útil para especificar sistemas de control de tráfico. Se han publicado varios modelos para simulación de tráfico basados en dicho formalismo, pero la mayoría de ellos intentan representar únicamente aspectos simples del tráfico de vehículos, no pudiendo ser analizados los problemas reales existentes.

En [DW99] se define un lenguaje de alto nivel, llamado ATLAS (Advanced Traffic Languaje Specifications), utilizado para definir modelos de tráfico en una ciudad, y que puede ser mapeado a los formalismos DEVS y Cell-Devs.

El objetivo principal de este trabajo es agregar a los modelos existentes aspectos no considerados que acercan la simulación al comportamiento real del tráfico, extendiendo el lenguaje de especificación con los mismos.
Los nuevos aspectos a modelar son los siguientes:

- *Especificación de ruteo de vehículos:* Se agregarán al modelo información de origen – destino de cada vehículo, determinando los posibles caminos a seguir. De esta forma, si se tiene información estadística sobre puntos de concentración urbana en una ciudad, se puede modelar el flujo real de vehículos y analizar, por ejemplo, alternativas a implementar en los caminos más transitados.

- *Monitoreo de congestión:* Los nuevos modelos mantendrán información de congestión permitiendo modelar el comportamiento de los vehículos en diferentes situaciones. Además, posibilita analizar la conveniencia de brindar información en tiempo real acerca del estado de los caminos.

- *Comportamiento del conductor:* Se incluirán características del conductor que describen su forma de manejo. De acuerdo a ellas los vehículos evidenciarán diferentes acciones ante los mismos eventos. Esto es importante, por ejemplo, para el agregado de señales o definición de nuevas normas de tránsito.

Las funcionalidades anteriormente mencionadas serán mapeadas a los formalismos DEVS y Cell-DEVS, teniendo en cuenta diferentes alternativas de implementación.

Otro objetivo del presente trabajo es analizar la utilidad y los alcances del lenguaje de especificación de alto nivel ATLAS, utilizándolo para describir en forma completa un sector real de ciudad. Además, este mismo sector será implementado utilizando una herramienta (N-CD++), que permite la simulación de sistemas que respeten los formalismos DEVs y Cell-DEVS.

El presente trabajo está organizado de la siguiente manera. El informe comienza con una introducción de la simulación de sistemas para luego introducir el formalismo DEVs y Cell-Devs y la herramienta N-CD++ (que permite la simulación de sistemas que respeten estos formalismos). Posteriormente se introduce el lenguaje de alto nivel ATLAS para la simulación de tráfico en una ciudad. Utilizando este lenguaje, se muestra además, la especificación completa de un sector real de ciudad y su respectiva implementación en la herramienta N-CD++.

En secciones subsiguientes se extiende el lenguaje de especificación ATLAS, agregando aspectos no considerados que acercan la simulación al comportamiento real del tráfico: incorporación de ruteo de vehículos, control de congestión y comportamiento del conductor. Además se incluye la especificación y la implementación de ejemplos varios, en donde se ilustra el uso de estas nuevas extensiones al lenguaje.

Sección II: Marco Teórico

Sección 2.1: Introducción a la Simulación

Un *sistema* es una entidad real o artificial que representa una parte de una realidad y está restringida por un entorno. Puede decirse que un sistema es un conjunto ordenado de objetos lógicamente relacionados que atraviesan actividades, interactuando para cumplir los objetivos propuestos [Wai98].

Un *modelo* es una representación inteligible (abstracta y consistente) de un sistema. En muchos casos no es posible la experimentación directamente sobre el sistema a estudiar, o se desea evitar costos, peligro, etc; por lo que se recurre al uso de modelos. Se distinguen dos grandes grupos de métodos para modelar sistemas complejos:

- **Analíticos:** Están basados en el razonamiento deductivo y permiten obtener soluciones generales al problema Un formalismo analítico muy difundido para el modelado de problemas es el uso de ecuaciones diferenciales. Sin embargo, los sistemas del mundo real generalmente son muy complejos, para los cuales se hace muy difícil, y en ciertos casos imposible, hallar soluciones analíticas o heurísticas con resultados satisfactorios. Para el estudio de este tipo de sistemas se ha difundido el uso de metodologías y herramientas de simulación

- **Basados en simulación :** en ellos no existen soluciones generales. Buscan soluciones particulares para el problema.

Giambiasi [Gia96] define **Simulación** como *"La reproducción del comportamiento dinámico de un sistema real en base a un modelo, con el fin de llegar a conclusiones aplicables al mundo real"*. Por ende, la simulación es el proceso de diseñar un modelo de un sistema real, y conducir experimentos basados en computadoras para describir, explicar y predecir el comportamiento del sistema real [W96].

Las ventajas de la simulación son múltiples: permite experimentación controlada, puede reducirse el tiempo de desarrollo del sistema, las decisiones pueden verificarse artificialmente, un mismo modelo puede usarse muchas veces, etc. Otra gran ventaja es que su uso no afecta al sistema real que modela, que puede seguir utilizándose (o no existir).

Algunos problemas que existen en el uso de simulación son su tiempo de desarrollo, en que los resultados pueden tener divergencia con la realidad (precisan validación), y en que para reproducir el comportamiento del sistema simulado se precisa colección extensiva de datos [W96]. Debido a esto, si el problema es simple, es conveniente el uso de métodos analíticos ya que permiten obtener soluciones generales. En cambio, para modelos complejos, usando simulación se pueden probar distintas condiciones de entrada para los cuales no serían posibles de probar analíticamente, y obtener resultados de salida significativos.

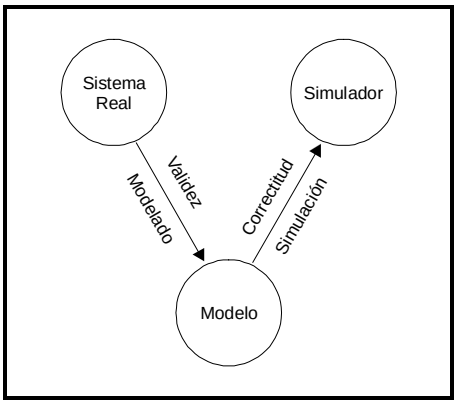


Figura 1 – Relaciones de modelado y simulación [Zei90]

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 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.

Sección 2.2: Autómatas Celulares

Los autómatas Celulares son un formalismo para modelado de sistemas dinámicos, complejos, de variables y tiempos discretos (el tiempo, espacio, y estados del sistema son discretos), creados originalmente por Von Neuman y S. Ulam. Estos se definen como un conjunto infinito n-dimensional de celdas ubicadas geométricamente que conforman una grilla o malla de celdas. Cada punto de la grilla puede tener un estado elegido de un alfabeto infinito. Cada una contiene el mismo aparato de cálculo que las otras y se conectan entre si de forma uniforme. El estado de las celdas se actualiza simultáneamente y de forma independiente de las demás en pasos de tiempo discreto [Wol86], utilizando una función de computo local. Esta función considera el estado de la celda actual y de un grupo de celdas cercanas, al que se denomina *vecindad de una celda*. Esta vecindad es homogénea para todas las celdas.

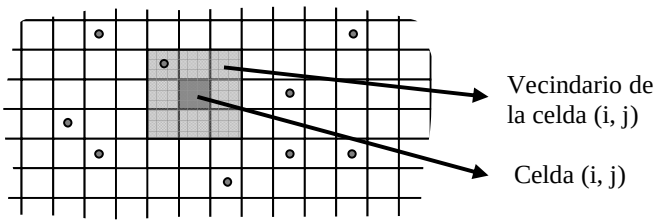


Figura 2 – Fragmento de un Autómata Celular Bidimensional

Cada celda se define como un modelo atómico, y se incluye un procedimiento para acoplar celdas. Se permite, además, la inclusión de demoras de transporte y demoras inerciales.

Los autómatas celulares suelen usarse, al igual que las ecuaciones diferenciales, para el modelado de sistemas físicos. Existen autómatas para modelar dinámica en fluidos y gases, colonias de animales, tránsito vehicular, modelos ecológicos y problemas de criptografía, entre otros. 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.

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.

Un ***autómata celular conceptual*** puede definirse como:

$$CCA = < S, n, C, \eta, N, T, \tau, c.Z_0^+ >$$

donde

S es el alfabeto utilizado para representar el estado de cada celda
n es la dimensión del espacio de celdas
C es la definición del conjunto de estados para el espacio de celdas
η es el tamaño de la vecindad
N es el conjunto de vecindad
T es la función de transición global
τ es la función de cómputo local y
$c.Z_0^+$ es la base de tiempo (discreta) del autómata celular.

Analicemos con detalle la definición de cada uno de estos conjuntos:

$\mathbf{S} \subseteq Z \wedge \#S < \infty$

$\mathbf{n} \in N$

Notación 1
De aquí en más, se llamará $C_c \in S$ al estado de la celda c, siendo $c \in \mathbf{Z}^n$, $c = (i_1,...,i_n)$ la posición de la celda c en el espacio n-dimensional. Aquí, $\forall\, k \in [1,n]$, $i_k \in \mathbf{Z}$ es la posición de la celda en la k-ésima dimensión.

Para el caso de autómatas celulares conceptuales, $\forall\, k \in [1,n]$, $i_k \in [-\infty,\infty]$.

$\mathbf{C} = \{ C_c / c \in Z^n \wedge C_c \in S \}$

Para el caso de autómatas celulares bidimensionales [WG98b],

$C = \{ C_{ij} / i,j \in \mathbf{Z} \wedge C_{ij} \in S \}$

$\boldsymbol{\eta} \in \mathbf{N}$

$\mathbf{N} = \{ (v_{k1},...,v_{kn}) / \forall\, (k \in \mathbf{N}, k \in [1, \eta]) \wedge (i \in \mathbf{N}, i \in [1, n]), v_{ki} \in \mathbf{Z} \}$

$\mathbf{T}$: $C \times c.Z_0^+ \rightarrow C$

$\boldsymbol{\tau}$: $N_c \times c.Z_0^+ \rightarrow C_c$ $\forall\, c \in \mathbf{Z}^n$. En este caso, $C_c[t+c] = \tau(C_{c+v1}[t],...,C_{c+v\eta}[t])$, donde $t \in c.Z_0^+ \wedge$
$\forall\, (k \in \mathbf{N}, k \in [1, \eta]), vk \in N \wedge c+vk = (i_1+v_{k1}, ..., i_n+v_{kn})$.

Para autómatas celulares de dos dimensiones, la función de cálculo local [WG98b] puede definirse como:

$\boldsymbol{\tau}$: $S^\eta \times c.Z_0^+ \rightarrow S$. En este caso, $C_{ij}[t+c] = \tau(C_{i0,j0}[t],..., C_{i\eta, j\eta}[t])$,
con $(ik = i+v_{ki} \wedge jk = j+v_{kj})$ $\forall\, (i, j \in \mathbf{Z}) \wedge ((v_{ki}, v_{kj}) \in N, k \in \mathbf{N}, k \in [1, \eta])$.

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

Como podemos ver en la definición, el modelo está compuesto de un espacio **n**-dimensional (**C**) de celdas. El espacio de estados progresa en pasos de tiempo discreto, dado que la base de tiempo discreta está definida por $c.\boldsymbol{Z}_0^+$ (un conjunto de valores enteros separados entre sí por una constante de tiempo).

En este caso se incluyen dos definiciones: una considera espacios de n dimensiones en los cuales los vecinos son homogéneos (pero pueden pertenecer a cualquier parte del espacio de celdas). Por otro lado, se incluyen definiciones para autómatas celulares de dos dimensiones con vecindades adyacentes a la celda origen.

El estado de cada celda en el espacio puede tomar un valor entre los de un alfabeto finito (**S**). El dominio del conjunto S puede ser cualquier conjunto finito discreto. De aquí en más se considerará esta definición presentada para el conjunto S ($\mathbf{S} \subseteq Z \wedge \#S < \infty$), sin pérdida de generalidad en la definición del alfabeto de la celda. Esto se debe a que, a pesar de que puede usarse cualquier conjunto finito de símbolos, se puede hacer un mapeo de estos en el S definido en esta sección.

La vecindad está definida como una lista de $\boldsymbol{\eta}$ vecinos de n dimensiones. En la definición se usa un índice (k) que permite identificar el número de vecino, y un segundo índice (i) que indica la dimensión de una de las posiciones del vecino. En este caso, los vecinos son una n-upla de posiciones relativas a la celda origen.

Como vemos, las vecindades consideradas implican autómatas celulares homogéneos, aunque las definiciones pueden extenderse fácilmente para autómatas inhomogéneos. En este caso, N se debe definir como un arreglo de listas de vecindades, especificado formalmente como:

$$N = \{N_c / c \in \boldsymbol{Z}^n\}$$

con

$$N_c = \{ (v_{k1},...,v_{kn})_c / \forall (k \in \boldsymbol{N}, k \in [1, \eta_c]) \wedge (i \in \boldsymbol{N}, i \in [1, n]), v_{ki} \in \boldsymbol{Z} \}$$

Si suponemos el caso de espacios bidimensionales,

$$N = \{N_{kl} / k, l \in \boldsymbol{Z}\}$$

donde

$$N_{kl} = \{(i_p,j_p) / \forall p \in \boldsymbol{N}, p \in [1,\eta_{kl}], i_p, j_p \in \boldsymbol{Z} \wedge \eta_{kl} \in \boldsymbol{N}\}$$

En este caso, cada celda tendrá una lista de vecinos de η_c (η_{kl}) elementos, compuesta por tuplas de índices relativos a la celda de origen.

La evolución del espacio de estados del autómata está definida por la ejecución de una función de transición global (**T**) que cambia el estado de todo el espacio de celdas. El comportamiento de esta función de transición depende del resultado de la ejecución de funciones de cálculo locales ($\boldsymbol{\tau}$) que corren en el entorno de la vecindad de una celda (**N**). Conceptualmente, el cálculo de estas funciones locales se hace de forma sincrónica y paralela en cada celda del espacio. En el caso de autómatas celulares bidimensionales, $C_{ij}[t]$ es el estado de la celda (i,j) en el tiempo simulado t, y τ denota la función de cálculo local para el autómata. Los parámetros (i0,j0),..., (iη,jη) definen la vecindad de la celda.

En el caso anterior, los índices del espacio de celdas podían incluir infinitas celdas. Como fuera comentado previamente, estamos interesados en modelos ejecutables. Para lo cual un autómata

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

donde

S es el alfabeto utilizado para representar el estado de cada celda
n es la dimensión del espacio de celdas
$\{t_1,...,t_n\}$ es la cantidad de celdas en cada una de las dimensiones
C es la definición del conjunto de estados para el espacio de celdas
η es el tamaño de la vecindad
N es el conjunto de vecindad
T es la función de transición global
B es el conjunto de celdas de borde
τ es la función de cómputo local y
$c.Z_0^+$ es la base de tiempo.

celular ejecutable sincrónico se define formalmente como:

En este caso,

$S \subseteq Z \wedge \#S < \infty$

$n \in N$; $n < \infty$

$\{t_1,...,t_n\} \in N$, con $t_k < \infty \; \forall \; k \in [1,n]$

$C = \{ C_c / c \in I \wedge C_c \in S \}$, con $I = \{ (i_1,...,i_n) / i_k \in N \wedge i_k \in [1, t_k] \; \forall \; k \in [1, n] \}$ (1)

Para el caso de autómatas celulares bidimensionales [WG98b], se obtiene
$C = \{ C_{ij} / i,j \in N \wedge i \in [1,f], j \in [1,c] \}$, donde $t_1 = f$ es la cantidad de filas y $t_2 = c$ es la cantidad de columnas

$\eta \in N$ es el tamaño de la vecindad

$N = \{ (v_{k1},...,v_{kn}) / \forall (k \in N, k \in [1, \eta]) \wedge (i \in N, i \in [1, n]), v_{ki} \in Z \wedge t_i - | v_{ki} | \geq 0 \}$

Para el caso de autómatas celulares bidimensionales con vecindades adyacentes a la celda de origen [WG98b],
$N = \{ (i_p,j_p) / i_p, j_p \in Z \wedge i_p, j_p \in [-1, 1] \; \forall \; p \in N, p \in [1,\eta] \}$.

B = $\{\varnothing\}$ si el espacio de celdas es toroidal; o
B = $\{C_b / C_b \in C$ con $b \in I \}$, con **I** definido como en (1). En este caso, el conjunto B está sujeto a la restricción que $\tau_b \neq \tau_c = \tau \; \forall \; (C_c \notin B) \wedge (C_b \in B)$.

En general, si $B \neq \{\varnothing\}$, B suele definirse como:

$B = \{C_b / C_b \in C$ 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 $\tau_b \neq \tau_c = \tau \; \forall \; c \notin L$.

Para el caso de espacios bidimensionales [WG98b] se obtiene:

B = {∅} si el espacio de celdas es toroidal; o
B = {C_{ij} / $C_{ij} \in C \wedge (i = 1 \vee i = f) \wedge (j = 1 \vee j = c)$ } sino.

T: C x $c.\boldsymbol{Z}_0^+ \rightarrow$ C;

τ: N_c x $c.Z_0^+ \rightarrow C_c$; donde $C_c[t+c] = \tau(C_{c+v1}[t],...,C_{c+v\eta}[t])$, (2)

con $t \in c.Z_0^+ \wedge \forall (k \in \boldsymbol{N}, k \in [1, \eta])$, $vk \in N \wedge c+vk = (i_1+v_{k1} \bmod(t_1), ..., i_n+v_{kn} \bmod(t_n))$ en el caso que B = {∅}, y

$C_b[t+c] = \tau_b(C_b[t]), \forall b \in B$, con $t \in c.Z_0^+$. En este caso, $\tau_b \neq \tau_c = \tau \; \forall c \notin B$, y para estos últimos, C_c se calcula como en (2).

Como puede verse, la definición de los autómatas celulares ejecutables difiere en algunos aspectos de la de los autómatas celulares conceptuales. Por un lado, se acota el tamaño del espacio de celdas en cada una de las dimensiones (**t_1,...,t_n**), que en este caso también es finita. Por ende, los subíndices de las celdas también se acotan (números naturales finitos).

Otra restricción de los autómatas ejecutables es la pérdida de homogeneidad del espacio de celdas. Ello implica la inclusión de un conjunto de celdas de borde (**B**), que puede ser vacío (para los autómatas celulares toroidales), o un conjunto de celdas prefijado. Estas celdas tienen distinto comportamiento que el resto, por ende, usan distintas funciones de transición local, que se calculan de forma diferente en los bordes y en el resto del autómata.

Como en el caso de los autómatas celulares conceptuales, la semántica de la función de transición T implica la ejecución simultánea y paralela de las funciones de transición local en todo el autómata celular. Esta puede plantearse formalmente como:

$$C_c \in C \;\; \forall c = \{ (i_1,..., i_n) / i_j \in [1, t_j] \; \forall j \in [1,n] \}, \qquad t \in c.\boldsymbol{Z}_0^+$$

$$C[t+c] = T(C[t]), \qquad \text{con } C_c[t+c] = \tau(C_c[t]) \; \forall c = \{ (i_1,..., i_n) / i_j \in [1, t_j] \; \forall j \in [1,n] \}$$

Finalmente, para el caso de ***autómatas celulares asincrónicos***, se puede considerar la siguiente definición formal:

***ACA* = < *S, n, {t_1,...,t_n}, C, η, N, B, Nevs, T, τ, R_0^+* >**

donde

S es el alfabeto utilizado para representar el estado de cada celda;
n es la dimensión del espacio de celdas;
{**t_1,...,t_n**} es la cantidad de celdas en cada una de las dimensiones;
C es la definición del conjunto de estados para el espacio de celdas;
η es el tamaño de la vecindad;
N es el conjunto de vecindad;
B es el conjunto de celdas de borde;
Nevs es una lista de próximos eventos;
T es la función de transición global;
τ es la función de cómputo local; y
$\boldsymbol{R}_0^+$ es la base de tiempo (continua) del autómata celular.

En este caso,

$\mathbf{S} \subseteq \boldsymbol{Z} \wedge \#S < \infty$

$\mathbf{n} \in \boldsymbol{N}$; $n < \infty$

$\{\mathbf{t_1},...,\mathbf{t_n}\} \in \boldsymbol{N}$, siendo $t_k < \infty \; \forall \; k \in [1,n]$

$\mathbf{C} = \{ C_c / c \in \mathbf{I} \wedge C_c \in S \}$, con **I** definido como en (1)

$\boldsymbol{\eta} \in \boldsymbol{N}$

$\mathbf{N} = \{ (v_{k1},...,v_{kn}) / \forall (k \in \boldsymbol{N}, k \in [1, \eta]) \wedge (i \in \boldsymbol{N}, i \in [1, n]), v_{ki} \in \boldsymbol{Z} \wedge t_i - | v_{ki} | \geq 0 \}$

B = {∅} si el espacio de celdas es toroidal o
B = $\{C_b / C_b \in C$ con $b \in \mathbf{I} \}$, con **I** definido como en (1). En este caso, el conjunto B está sujeto a la restricción que $\tau_b \neq \tau_c = \tau \; \forall \; (C_c \notin B) \wedge (C_b \in B)$.

Nevs = $\{ (c, t) / c \in \mathbf{I} \wedge t \in \boldsymbol{R_0}^+ \}$, donde c es la posición de una celda en el espacio, **I** está definido como en **I** = $\{ (i1,...,in) / ik \in \boldsymbol{N} \wedge ik \in [1, tk] \; \forall \; k \in [1, n] \}$ (1), y t es la hora del evento correspondiente.

T: C x $\boldsymbol{R_0}^+ \rightarrow$ C

τ: N_c x $R_0^+ \rightarrow C_c$; donde $C_c[t_p] = \tau(C_{c+v1}[t],...,C_{c+v\eta}[t])$, (3)
con $t \in \boldsymbol{R_0}^+ \wedge \forall (k \in \boldsymbol{N}, k \in [1, \eta])$, $vk \in N \wedge c+vk = (i_1+v_{k1} \bmod(t_1), ..., i_n+v_{kn} \bmod(t_n))$ en el caso que B = {∅}, y
$C_b[t_p] = \tau_b(C_b[t])$, $\forall$ b ∈ B, con $t \in \boldsymbol{R_0}^+$. En este caso, $\tau_b \neq \tau_c = \tau \; \forall \; c \notin B$, y para estos últimos, C_c se calcula como en (3). Aquí, $t_p = \min\{t_i\}_{i=1}^{n}$, con $i, p \in \boldsymbol{N} / \; t_i \in \boldsymbol{R_0}^+$. En este caso, $b = c_p \vee c = c_p$, con $(t_i, c_i) \in$ Nevs con la restricción que $\tau_b \neq \tau_c = \tau \; \forall \; c \notin \mathbf{I}$ (1), y para estos últimos, C_c se calcula como en (3).

Como puede verse, la mayoría de los conjuntos y funciones son similares que para el caso sincrónico. Los cambios se deben a que aquí, la base de tiempo es continua (es decir, que las variables de tiempo $t \in \boldsymbol{R_0}^+$). Esta definición asincrónica provoca que el formalismo deba incluir una lista de próximos eventos (**Nevs**), que debe utilizarse para almacenar la información correspondiente a los eventos a ser tratados.

En este caso, la semántica de la función de cálculo global (**T**) es diferente al caso anterior. Aquí, esta función implica la ejecución un grupo de celdas no latentes, llamadas inminentes. La ejecución de esta función se hace simultáneamente en todas las celdas inminentes para un tiempo simulado dado. Esto puede especificarse como:

$$C_c \in C \; \forall \; c \in \mathbf{I} \; (1), \quad t_p = \min\{t_i\}_{i=1}^{n}, \text{ con } (t_i, c_i) \in \text{Nevs} \Rightarrow C_{p=} \{ (t_p, c_p) / (t_p, c_p) \in \text{Nevs} \}$$

$$C[t_p] = T(C[t]), \quad \text{con } C_{cp}[t_p] = \tau(C_{cp}[t]) \; \forall \; c_p \in C_p \quad \wedge \quad C_c[t_p] = C_c[t] \; \forall \; c \notin C_p$$

Esta definición puede ser extendida para que el conjunto de estados posibles, S, sea un conjunto de conjuntos de números. Es decir, se representa el estado de una celda como un conjunto de números, que podrían ser enteros o reales. Esto es necesario para permitir el modelado de los sistemas con variables inherentemente reales y de aquellos cuyo estado queda mejor definido con más de sola variable entera. Para estos casos se utilizará la notación que se define a continuación.

Notación 2 :

En la descripción de los autómatas celulares para los modelos de la siguientes secciones, se notará como $\mathbf{C_{k}.s_i}$, al valor de la componente s_i del estado de la k-ésima celda; cuando el estado de una celda está compuesto por un conjunto de valores enteros, es decir, $\mathbf{S} = \{ (s_1, s_2, ..., s_n) / s_i \in Z \text{ para todo } i \in [1,..,n] \}$.

Sección 2.3: Aplicación de Autómatas Celulares a la simulación de tráfico

Un área de aplicación del formalismo de los Autómatas Celulares es la simulación de sistemas de control de tráfico de vehículos. El trafico de vehículos presenta algunas de las características de un sistema complejo: estados estables e inestables, comportamiento determinístico, caótico y estocástico.

A través de la simulación es posible analizar el impacto que tendrá un evento o la introducción de un nuevo elemento en el sistema de tráfico de una ciudad. Como ejemplo podemos citar:

- Determinación del sentido de circulación de las calles.
- Definición de nuevas normas de tránsito (ej: prioridad de paso, límite de velocidad, establecimiento de carriles especiales, etc.).
- La instalación o sincronización de semáforos.
- Disponibilidad de información a través de carteles electrónicos sobre el estado de las rutas.
- Cortes o bloqueos parciales de calles, etc.

En [DW99] se presenta una taxonomía que permite clasificar modelos para simulación de tráfico.

Taxonomía

Un aspecto importante de los modelos de simulación de flujo de vehículos es la estructura elegida para representar las calles, autopistas o vías de circulación; ella determina en gran medida el tipo de movimiento que caracterizará a los vehículos. Esto permite distinguir entre modelos simples que sólo representan el flujo de vehículos sobre un único carril, y los más complejos que modelan varios carriles de distinta dirección y con cruces. Estos últimos, de acuerdo a su estructura, pueden hacer que los vehículos tengan un movimiento más completo; modelando por ejemplo, giros y cambios de carril.

Otro aspecto importante que hay que considerar al simular tráfico, son las características específicas que se desean modelar que afectan el movimiento de los vehículos, a parte de la estructura de las calles. Esto puede incluir la representación de diversos tipos de vehículos (autos, camiones, colectivos, taxis, etc.), señales de tránsito, desvíos, choques, etc.

Teniendo en cuenta los aspectos mencionados se definen dos clasificaciones para modelos de simulación de tráfico, en la primera se utiliza como criterio la estructura de las calles y en la segunda las características adicionales que afectan el movimiento de los vehículos. Estas últimas son independientes de la estructura de las calles, es decir, pueden estar presentes o no en cualquiera de las subclases determinadas por la primera; este fue el motivo principal para definir dos clasificaciones en vez de una sola. Por lo tanto, ambas son complementarias y utilizadas en conjunto permiten especificar una mayor cantidad de características del modelo.

En definitiva, las clasificaciones resultantes para analizar modelos existentes de autómatas celulares para simulación del tráfico son definidas como [DW99]:

1) Clasificación según la estructura de las calles

A. Modelos de tráfico de carril único
   A.1. Modelos de una dimensión (sin cruces)
   A.2. Modelos de dos dimensiones (con cruces)
B. Modelos de tráfico de dos carriles
   B.1. Modelos con una única dirección
      B.1.1. Modelos con reglas simétricas
         B.1.1.1. Modelos sin cruce de carriles
         B.1.1.2. Modelos con cruce de carriles
      B.1.2. Modelos con reglas asimétricas
         B.1.2.1. Modelos sin cruce de carriles
         B.1.2.2. Modelos con cruce de carriles
   B.2. Modelos con dos direcciones
      B.2.1. Modelos sin cruces de carriles
      B.2.2. Modelos con cruces de carriles
C. Modelos de tráfico de más de dos carriles
   C.1. Modelos con una única dirección
      C.1.1. Modelos con reglas simétricas
         C.1.1.1. Modelos sin cruces de carriles
         C.1.1.2. Modelos con cruces de carriles
      C.1.2. Modelos con reglas asimétricas
         C.1.2.1. Modelos sin cruces de carriles
         C.1.2.2. Modelos con cruces de carriles
   C.2. Modelos con dos direcciones
      C.2.1. Modelos con reglas simétricas
         C.2.1.1. Modelos sin cruces de carriles

C.2.1.2. Modelos con cruces de carriles
C.2.2. Modelos con reglas asimétricas
C.2.2.1. Modelos sin cruces de carriles
C.2.2.2. Modelos con cruces de carriles

En los modelos de carril único el tráfico se representa como una única línea con flujo de vehículos. Luego pueden permitir o no que estos carriles independientes se crucen para formar las esquinas, constituyendo los modelos de dos dimensiones (con cruces) y de una dimensión (sin cruces), respectivamente.

Los modelos de tráfico de dos carriles se representan como dos líneas con flujo de vehículos paralelas. A su vez estos modelos pueden permitir que los dos carriles tengan la misma o distinta dirección de circulación. En el primer caso, el movimiento de cambio de carril puede definirse con reglas simétricas o asimétricas. Las reglas son simétricas si las condiciones para que los vehículos pasen de un carril a otro, son las mismas; caso contrario son asimétricas y se utilizan cuando por ejemplo un carril se utiliza sólo para adelantarse.

Los modelos de tráfico de más de dos carriles se representan como más de dos líneas con flujo de vehículos paralelas. Estos modelos, por lo general, son generalizaciones de los de dos carriles y se pueden clasificar considerando los mismo aspectos. La única diferencia es que dentro de los modelos de 2 direcciones (C.2), existe también la posibilidad de definir reglas simétricas o asimétricas para los carriles de igual dirección.

Cada modelo de la clasificación anterior, a su vez puede representar algunas de las siguientes características, o bien, una combinación de ellas,

2) Clasificación según características específicas a modelar

I. Vehículos especiales: en este caso se modelan vehículos con alguna característica o comportamiento diferente. Entre ellos se pueden considerar los siguientes:
1.1. Autos
1.2. Camiones
1.3. Colectivos
1.4. Ambulancias
1.5. Taxis
1.6. Bicicletas
1.7. Motos
1.8. Vehículos con ruta prefijada

2. Controles sobre el tráfico: en este caso los vehículos deben respetar las normas de tránsito incluidas en el modelo, que pueden ser:
2.1. Semáforos (con/sin flecha de giro)
2.2. Señales (PARE, Escuela, No Adelantarse, Ceda el Paso, Velocidad Máxima, Velocidad Mínima, Cruce de Peatones, Zona de Animales Sueltos, etc.)
2.3. Cruces de trenes
2.3.1. Con barrera
2.3.2. Sin barrera
2.3.3. Sólo el tren, sin cruce

2.4. Carriles con prioridades fijas, como por ejemplo, los autos que llegan por la derecha a un cruce tienen prioridad sobre los otros
2.5. Carriles especiales (por ejemplo, exclusivos para taxis y colectivos)
2.6. Bocacalles
2.7. Elevaciones transversales (lomas de burro)
2.8. Depresiones transversales (badén)
2.9. Irregularidad continua (serrucho)

3. Comportamientos anómalos de los vehículos: que pueden incluir,
   3.1. Autos con movimiento erráticos (zig zag)
   3.2. Autos que circulan por el medio entre dos carriles
   3.3. Autos maniobrando para estacionar
   3.4. Autos detenidos
   3.5. Autos circulando marcha atrás

4. Obstáculos en la calle:
   4.1. Obras (hombres trabajando)
   4.2. Baches
   4.3. Choques y otros accidentes
   4.4. Autos mal estacionados
      4.4.1. En doble mano
      4.4.2. Sobre la mano izquierda
   4.5. Peatones:
      4.5.1. Descuidados: cruzan sin mirar si se acercan autos
      4.5.2. Precavidos: cruzan prestando atención al tráfico (considerando autos que doblan o autos que siguen derecho, con o sin semáforo).

Seccion 2.4: Formalismo DEVS

Actualmente, para gran variedad de aplicaciones complejas se usan modelos y simulación. En algunos de estos sistemas las variables son discretas a tiempo continuo. Tales sistemas reciben el nombre de ***Sistemas Dinámicos de Eventos Discretos*** (**DEDS** – *Discrete Events Dynamic Systems*) en oposición a los **Sistemas Dinámicos de Variables Continuas** (**CVDS** - *Continuous Variable Dynamic Systems*) que se describen por ecuaciones diferenciales [Wai98]. Un paradigma de simulación para DEDS asume que el sistema simulado sólo cambia de estado en puntos discretos del tiempo, ante la ocurrencia de un evento. Un **evento** es un cambio de estado que debe efectuarse a una hora ti $\in$ $\boldsymbol{R}$ (el tiempo es continuo).

Zeigler [Zei76] propuso un formalismo conocido como **DEVS** (*Discrete EVents Systems specifications*) que permite la construcción jerárquica de modelos de eventos discretos con tiempo continuo. Esto favorece la creación de los mismos, ya que pueden ser considerados como nuevos componentes para ser usados en la creación de otros modelos, reduciendo así los tiempos de desarrollo y prueba. Además de un medio para construir modelos simulables, provee una representación formal para manipular matemáticamente sistemas de eventos discretos. El

formalismo define cómo generar nuevos valores para las variables y los momentos en los que estos valores deben cambiar.

DEVS es un formalismo universal para modelar y simular DEDS. Puede verse como una forma de especificar sistemas cuyas entradas, estados y salidas son constantes en intervalos, y cuyas transiciones se identifican como eventos discretos. Los intervalos de tiempo entre ocurrencias son variables [Wai98].

Un modelo DEVS se construye sobre la base a un conjunto a modelos básicos llamados Atómicos, que se combinan para formar modelos acoplados y un conjunto de relaciones que indican como los modelos están conectados en forma jerárquica.

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 los modelos de mayor nivel [W96].

La simulación es llevada a cabo por un *simulador abstracto*, que es genérico. En el concepto de simulador abstracto, la simulación de los modelos atómicos y acoplados es realizada por distintos procesadores llamados simuladores y coordinadores. Básicamente la simulación es disparada por la recepción de mensajes de simulación que invocan secuencialmente a las salidas, transiciones externas, y transiciones internas planificadas por las funciones de tiempo. En esencia, un simulador abstracto es una descripción algorítmica de como llevar a cabo las instrucciones implícitas en los modelos DEVS para generar su comportamiento. La simulación se efectúa por pasaje de mensajes entre los distintos procesadores que pueden acarrear información con respecto a eventos internos y externos, así como necesidades de sincronización [W96].

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 $j \in \boldsymbol{N}$, $j \in [1, \mu]$, $(\mu \in \boldsymbol{N}, \mu < \infty)$, 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.

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:
$P_j^i = \{ (N_j^i, T_j^i) \ / \ \forall\, j \in [1, \mu], (\mu \in \boldsymbol{N}, \mu < \infty), N_j^i \in [i_1, i_\mu]$ (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.

Sección 2.5: Formalismo Cell - DEVS

Cell_DEVS es un paradigma de especificación utilizado para describir modelos celulares en forma de modelos atómicos DEVS con distintos tipos de demoras. Es decir, cada celda será definida como un modelo atómico DEVS que podrá tener asociada algún tipo de demora. Luego estas celdas serán acopladas para formar un espacio Cell_DEVS completo a través de la relación de vecindad. Estos espacios a su vez pueden ser acoplados con otros espacios Cell_DEVS con distinta definición de la vecindad, con distinto comportamiento, o bien, con otros modelos de la jerarquía DEVS. Cada celda en el modelo puede tener dos tipos de demora: demora de transporte o demora inercial.

Las demoras de transporte permiten representar un comportamiento de actualización de estados secuencial con respecto a la llegada de los eventos a cada celda, es decir, posee un procesamiento FIFO registrando todos los valores en caso de un cambio de estado.

La demora inercial es muy útil cuando se desea representar una semántica con remoción para el comportamiento de una celda. La definición para las celdas que poseen demora inercial es idéntica a la dada para las celdas con demora de transporte. La diferencia radica en el tratamiento de los eventos externos e internos y la estructura de la celda.

Una celda con demora inercial tiene por política el desalojo o descarte de los valores. Por lo tanto, sirve para representar que una celda toma determinado estado sólo en el caso que no se produzca ningún evento desde el cálculo del nuevo estado hasta el tiempo especificado en la demora que provoque que el valor calculado para el nuevo estado sea modificado, es decir, la prioridad la tiene el último evento producido.

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

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

donde 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,
$phase \in \{activa, pasiva\}, \sigma \in \boldsymbol{R_0}^+ \cup \infty\}$

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

$\mathbf{d} \in \boldsymbol{R_0}^+, d < \infty$ es la demora de la celda

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

$\boldsymbol{\delta_{ext}}: Q \times X \rightarrow \theta$ 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

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

$\mathbf{D}: \theta \times N \times d \rightarrow \boldsymbol{R_0}^+ \cup \infty$, es la función de duración de vida del estado.

Aquí $\mathbf{D}(s, phase, \sigma_{queue}, \sigma, N, d) = t$ 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:

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

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 ***R***.
P^y es un port de salida que acepta valores de ***R***.

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

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

$\eta \in \boldsymbol{N}$, 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]. 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 el 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].

Sección 2.6: Herramienta N–CD++

N–CD++ es una herramienta de simulación que permite la creación de modelos Cell-DEVS n–dimensionales [RW99], donde el estado de cada celda pertenece al conjunto $\boldsymbol{R} \cup \{ \textbf{?} \}$ y **?** representa al valor *indefinido*. Incluye un lenguaje de especificación que permite describir el comportamiento de cada celda de un modelo celular. 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 construcción de un modelo completo que se desea simular puede especificarse en la herramienta describiendo uno a uno los componentes en forma jerárquica y su interrelación.
La posibilidad de generar los modelos en forma jerárquica es de mucha utilidad ya que esto permite la creación y prueba de los mismos en forma mucho más simple.

La herramienta consta de dos partes: un motor de simulación y los fuentes de los modelos atómicos que se desea puedan participar en alguna simulación.

El software de simulación provee el motor, junto con las estructuras que permiten configurar la especificación y asociación de los modelos que componen el sistema DEVS a simular.

Para determinar el comportamiento de las celdas, N-CD++ permite describir el comportamiento de cada una por medio de un lenguaje de expresiones lógicas a través del cual se define la función de transición local.

El tamaño del espacio celular, el tipo de delay que usará el modelo y si tiene borde o es circular son parámetros de la especificación del modelo.

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, indicando al modelador tal situación y 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.

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++* .

Sección 2.7: ATLAS (Advance Traffic Languaje Specifications)

En [DW99] se define un lenguaje de alto nivel para la especificación de secciones de ciudad que permite representar una gran variedad de características del tráfico urbano. Se cubren aspectos tales como circulación de autos y camiones, presencia de semáforos, choques, baches, etc. El objetivo de este lenguaje es proveer una herramienta para la definición de secciones de ciudades, brindando un nivel de abstracción entre la especificación del problema y los modelos que luego se definan para simular el comportamiento. Por lo tanto, el usuario del lenguaje no necesita interactuar con los modelos de simulación y por ende no es necesario que conozca los formalismos utilizados para especificarlos.

El lenguaje está formado por conjuntos que modelan cada una de las características del tráfico. Por ejemplo, los conjuntos de tramos y cruces, permiten diseñar el trazado de las calles; el conjunto de los semáforos señala las esquinas con este tipo de control, etc. Esto es, para diseñar el modelo de la ciudad, se deben definir los elementos de cada conjunto que representen las características particulares que se desean modelar. A continuación, se presenta un resumen de las construcciones del lenguaje, comenzando por el trazado de calles y siguiendo por diversos elementos de control que permiten aproximar en forma más completa el modelo a la realidad.

Una calle se representa como una secuencia de *tramos*, donde cada uno de éstos es una sección sin cruces y de mano única. Cabe destacar que un sector doble mano se define con un tramo en cada dirección. Cada tramo se especifica incorporando un elemento en el siguiente conjunto:

$$\text{Tramos} = \{(p1, p2, n, a, dir, max) / p1, p2 \in \text{Puntos} \wedge p1 \neq p2 \wedge n, max \in N \wedge a, dir \in \{0, 1\}\}$$

Por lo tanto, para cada tramo o elemento de este conjunto se deben identificar sus extremos (p1 y p2), la cantidad de carriles (n), si tendrá forma recta o curva (a), el sentido de circulación de los vehículos (dir) y finalmente, su velocidad máxima permitida (max).

Una vez definidas las calles, es útil identificar los cruces entre ellas, para poder más adelante definir diversos elementos que ejerzan control sobre el flujo de vehículos. Los *cruces* o intersecciones, se pueden obtener a partir de los tramos definidos, como:

$$\text{Cruces} = \{ (c, maxc) / maxc \in N \wedge \exists t, t' \in \text{Tramos} \wedge t = (p1, p2, n, a, dir, max) \wedge t' = (p1', p2', n', a', dir', max') \wedge t \neq t' \wedge (p1 = c \vee p2 = c) \wedge (p1' = c \vee p2' = c) \}$$

Los elementos de este conjunto son puntos del espacio de dos dimensiones que representan los lugares donde se cruzan 2 ó más tramos, asociados con su velocidad máxima de circulación permitida. Un cruce siempre debe tener por lo menos un tramo de ingreso y uno de salida, pues sino no es posible la circulación del tráfico.

Una vez definido el esqueleto de la ciudad, se puede modelar la presencia de diversos elementos de control que influyen sobre el tráfico de vehículos. Entre ellos, los *semáforos* se definen agregando elementos en el siguiente conjunto:

$$\text{CrucesSemáforos} = \{ c / c \in \text{Cruces} \}$$

Cada cruce de este conjunto representa una esquina con semáforos. Es decir, los vehículos que llegan a la intersección deben chequear el color del semáforo para determinar si pueden avanzar.

También se puede incorporar el trazado de las *vías del tren* mediante el siguiente conjunto:

$$RedDeVías = \{ Vías / Vías = \{ (t, \ell, seq) / t \in Tramos \wedge \ell \in N \wedge seq \in N \} \}$$

Cada elemento Vías ∈ RedDeVías representa el trazado de las vías de algún ramal de trenes. Para especificarlo se indican los lugares donde se ubican los pasos a nivel (intersección entre las vías y las calles donde se permite el cruce de los vehículos). Cada tupla, pn = (t, ℓ, seq), identifica la ubicación de un paso a nivel, es decir el tramo (t) y la distancia entre el comienzo del tramo y las vías (ℓ). Además indica el orden que le corresponde al paso a nivel (seq) para poder establecer la secuencia de avance del tren (en qué orden avanza sobre los pasos a nivel).

Las *obras* son secciones de calles deshabilitadas para la circulación de vehículos, debido a la presencia de obreros trabajando. Las obras se modelan incorporando elementos en el siguiente conjunto:

$$Obras = \{ (t, ni, \ell, \#n) / \ t \in Tramos \wedge t = (c1, c2, n, a, dir, max) \wedge ni \in [0, n-1] \wedge \ell \in N \wedge \#n \in [1, n+1-ni] \wedge \#n \equiv 1 \bmod 2 \}$$

Cada tupla del conjunto, o = (t, ni, ℓ, #n), identifica el tramo (t) donde se halla la obra, el primer carril (ni) afectado por la obra, la distancia sobre el carril ni que existe entre la columna central de la obra y el comienzo del tramo, y la cantidad de carriles que ocupa la obra (#n). Cada tupla especifica un rombo sobre un tramo por donde los vehículos no pueden circular. Por lo tanto sólo se pueden definir obras con esa forma, pero esto no implica una limitación en la expresividad del lenguaje porque en general son construidas así para permitir el desvío gradual del tráfico. Se pide que la cantidad de carriles sea impar para poder armar el rombo (si fuera par se obtiene un rombiode y no existe una celda central) y que el carril inicial (ni) más la cantidad de carriles que ocupa (#n) no sea más grande que la cantidad de carriles totales del tramo.

La presencia de *baches* en una calle se modela incorporando elementos en el siguiente conjunto:

$$Baches_T = \{ (t, n1, \ell) / t \in Tramos \wedge t = (c1, c2, n, a, dir, max) \ \wedge n1 \in [0, n-1] \wedge \ \ell \in N \}$$

Cada tupla del conjunto, b = (t, n1, ℓ), identifica el tramo (t) y el carril (n1) donde se encuentra el bache; y el desplazamiento del bache sobre el carril, es decir la distancia sobre el carril n1 que existe entre el bache y el comienzo del tramo (representada por ℓ).

Algunos *elementos de control* como las elevaciones transversales (lomo de burro), depresiones transversales (badén), bocacalles, irregularidades continuas (serrucho) y señales de PARE o de Escuela se definen en forma conjunta de la siguiente forma:

$$ElementosDeControl = \{ (t, e, \ell) / t \in Tramos \wedge \ell \in N \wedge e \in \{elevación\ transversal, depresión\ transversal, bocacalles, irregularidad\ continua, señal\ de\ PARE, señal\ de\ Escuela\} \}$$

Cada tupla del conjunto, ec = (t, e, ℓ), identifica el tramo (t = (c1, c2, n, a, dir, max)), el tipo de elemento de control se ha especificado (e) y la distancia entre el elemento de control y el comienzo del tramo (representada por ℓ).

También es posible definir calles con *carriles de estacionamiento*, incorporando tramos con esas características en el siguiente conjunto:

AutosEstacionados = { (t, n1) / t ∈ Tramos ∧ n1 ∈ {0, 1} ∧ t = (c1, c2, n, a, dir, max) ∧ n > 1 }

Cada tupla del conjunto, ce = (t, n1), identifica el tramo (t = (c1, c2, n, a, dir, max)) y el carril sobre el que estacionan los vehículos (representado por n1). Si n1 = 0, se estaciona sobre el carril 0 (carril izquierdo), si n1 = 1 lo hacen sobre el carril n-1 (carril derecho). Por lo tanto, el conjunto AutosEstacionados especifica calles con alguno o ambos bordes de estacionamiento e impone la restricción de que el tramo tenga por lo menos 2 carriles (n > 1), uno que es donde estacionan y el otro para la circulación de tráfico.

Para representar la circulación de *camiones* en el modelo de tráfico se definen nuevas calles y cruces que permitan diferenciarlos de los que son exclusivos para autos. Para definir un tramo con circulación de camiones se debe agregar un elemento en el siguiente conjunto:

TramosCamiones = { (p1, p2, n, a, dir, max) / p1, p2 ∈ Puntos ∧ n, max ∈ N ∧ a, dir ∈ {0, 1} }

Cada elemento de este conjunto es una tupla de 6 componentes, cuyo significado es el mismo que fuera definido para los tramos exclusivos para autos. La diferencia entre ambos se refleja en los modelos de comportamiento, que deberán reflejar la presencia de vehículos de tamaño más grande y de velocidad más lenta.

Para especificar *choques* de vehículos en los tramos, se debe indicar sobre cuáles se desea modelar este comportamiento, así se construye el siguiente conjunto:

TramosChoques = { t / t ∈ TramosCamiones }

Este conjunto contiene tramos en los que en algún momento de la simulación, en forma no determinística, se pueden producir los efectos de un choque. Esto es que en algunos sectores del tramo no se permita la circulación de vehículos durante un cierto tiempo.

De esta forma, es posible describir una sección de ciudad mediante la incorporación de elementos en los respectivos conjuntos, permitiendo realizar una definición incremental, que comience modelando sólo algunas calles y cruces en una primera instancia, y que luego se puedan ampliar con elementos de control o extendiendo la zona que se está planteando. En definitiva, utilizando este lenguaje, se obtiene un modelo de alto nivel que luego se puede mapear sobre diversos formalismos que permitan obtener una simulación del problema. En particular, en [DW99] se plantea un mapeo de este lenguaje sobre el paradigma Cell_DEVS [WG98].

Sección III: Aplicación de ATLAS para modelar un sector de ciudad

Atlas (Advance Traffic Languaje Specifications) es un lenguaje de especificación construido sobre los formalismos DEVS y CELL-DEVS, utilizado para describir secciones de ciudad[DW99].

En esta sección, presentamos el mapeo entre Atlas y modelos CELL-DEVS mediante un ejemplo. Se definirá el sector de ciudad que se muestra en la figura 1, especificando los modelos necesarios para representarlo. A su vez esta definición será utilizada como base para la implementación de este sector de ciudad utilizando la herramienta N-CD++ [RW99].

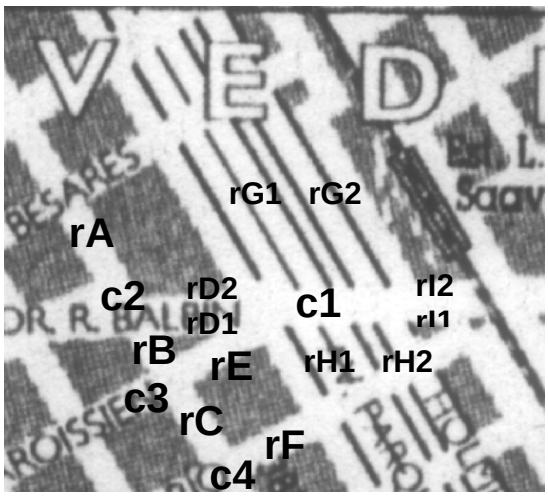


Figure 1 – Un sector de ciudad

En Atlas, un sector de ciudad se define a través de un conjunto de calles (tramos) conectados mediante cruces. En la figura 2 podemos ver en forma esquemática los tramos y cruces involucrados en el sector elegido.

Tramos

En la figura 1 observamos los siguientes tramos o rutas:

T = { rA, rB, rC, rD1, rD2, rE, rF, rG1, rG2, rH1, rH2, rI1, rI2}

Los mismos son definidos de la siguiente forma:

rA = [(0,0), (0,130), 1, 0, 1, 40]
rB = [(0,130), (0,200), 1, 0, 1, 40]
rC = [(0,200), (0,300), 1, 0, 1, 40]
rD1 = [(0,130), (100,200), 2 , 0, 1, 60]
rD2 = [(0,130), (100,200), 2, 0, 0, 60]
rE = [(0,200), (100,200), 1, 0, 0, 40]
rF = [(0,300), (100,300), 1, 0, 1, 40]
rG1 = [(100,0), (100,200), 4 , 0, 1, 60]
rG2 = [(100,0), (100,200), 4, 0, 0, 60]
rH1 = [(100,200), (100,300), 2, 0, 1, 60]
rH2 = [(100,200), (100,300), 2, 0, 0, 60]
rI1 = [(100,200), (200,280), 2, 0, 1, 60]
rI2 = [(100,200), (200,280), 2, 0, 0, 60]

Aquí se utilizó la definición presentada en la sección 1.1.1. de [DW99]. Como puede verse existen tramos con 1 (rA, rB, rC, rE y rF.), 2 (rD1, rD2, rH1, rH2, rI1, rI2) y 4 carriles (rG1, rG2). Este número es el que determina las dimensiones del Cell_DEVS asociado con el tramo y es importante para determinar el conjunto de reglas que representa el movimiento de los autos.

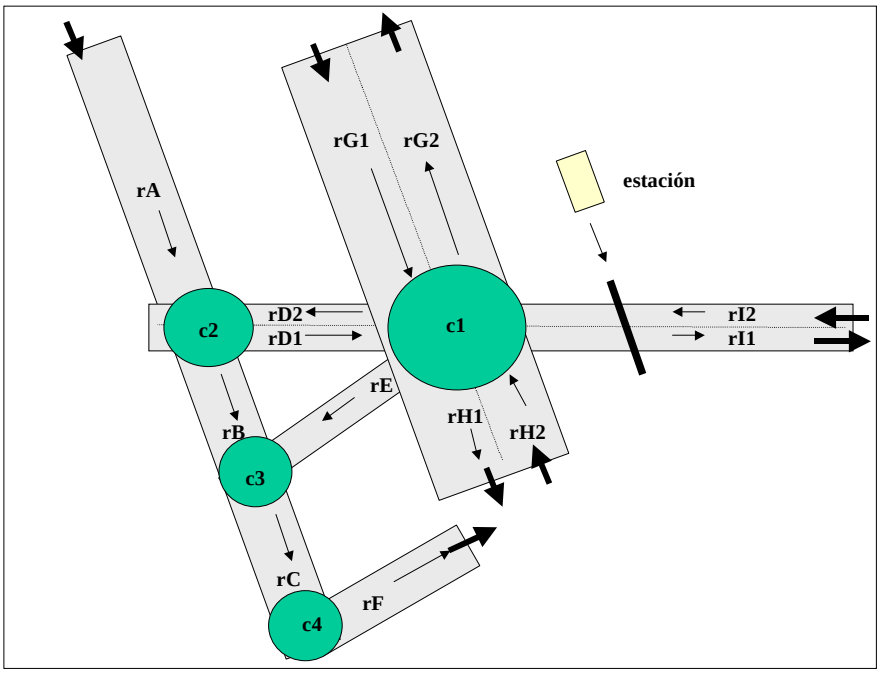


Figura 2 – Representación esquemática de tramos y cruces del sector de ciudad

Tramos con 1 carril

Como ejemplo de tramos de 1 carril será especificado el tramo rB. La especificación del resto de los tramos de 1 carril (rA, rC, rE y rF) es equivalente a la del tramo rB.

El tramo rB = [(0,130), (0,200), 1, 0, 1, 40] es mapeado con un Cell-DEVS de una dimensión con demora de transporte.

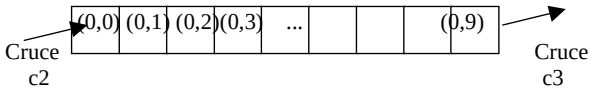


Figura 3. Tramo rB

El número de celdas del tramo (k) puede ser obtenido calculando la distancia entre los puntos (0,130) y (0,200), dividiendo el resultado por el tamaño de la celda. El tamaño de una celda es un parámetro de la simulación que representa aproximadamente el largo de un auto, ya que se asume que una celda es ocupada únicamente por un auto. Aquí, el valor de este parámetro es 7.5 m

(como en la mayoría de los modelos de autómatas celulares). De este modo k puede ser obtenida de la siguiente forma:

$$k = \left\lceil \sqrt{|x1 - x2|^2 + |y1 - y2|^2} \Big/ cell\_size \right\rceil = \left\lceil \sqrt{0 + 70^2} / 7.5 \right\rceil = 10$$

Esto significa que el tramo rB será representado por un espacio Cell-DEVS de una fila y diez columnas (i.e. n = 1 and $t_1 = 10$).

Cada celda en este espacio se define como:

$$C_{0j} = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

El vecindario de la celda se define como

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

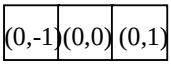


Figura 4. Vecindario de la celda

Las reglas que representan el movimiento de los autos(τ) , son aquellas definidas en [DW99] para tramos de un carril.

X = ***Y*** = {0, 1}
S:

s = { 1 si hay un vehículo en la celda
    { 0 sino.

delay = transport

d = Conversión_Demora(velocidad(40Km/h))

La demora de transporte es usada para modelar el tiempo que tarda un vehículo en abandonar una celda e ingresar a la siguiente. Esto depende de la velocidad actual del vehículo y es generada utilizando una función aleatoria.

El modelo acoplado correspondiente al tramo rB se define de la siguiente forma:

TC1(10, 40 Km/h) = < Xlist, Ylist, I, *X*, *Y*, n, $\{t_1,...,t_n\}$, η, N, C, B, Z, select >

Los parámetros 10 y 40 se utilizan para construir el modelo acoplado. La velocidad máxima es utilizada como parámetro para el generador de números aleatorios, y el parámetro k se obtiene computando la longitud del tramo (utilizando los límites del tramo y el tamaño de una celda).

n = 1; $\mathbf{t_1} = 10$
Ylist = **Xlist** = { (0,0), (0,9) }
I = <P^x, P^y>
P^x = {<$X_{\eta+1}$(0,0), binario>, <$X_{\eta+1}$(0,9), binario>}

$P^y = \{<Y_{\eta+1}(0,0), \text{binario}>, <Y_{\eta+1}(0,9), \text{binario}>\}$

Los ports se denominarán de la siguiente forma:

Port	Nombre
$X_{\eta+1}(0,0)$	x-c-auto
$X_{\eta+1}(0,9)$	x-c-haylugar
$Y_{\eta+1}(0,0)$	y-c-haylugar
$Y_{\eta+1}(0,9)$	y-c-auto

$\boldsymbol{X} = \boldsymbol{Y} = \{ 0, 1 \}$

$\eta = 3$; $\mathbf{N} = \{ (0,-1), (0,0), (0,1) \}$

$\mathbf{B} = \{ (0,0), (0,9) \}$

Las celdas (0,0) y (0, 9) conforman la interface externa del modelo. Cada una de ellas intercambia información con cruces (ver figura 5).
El comportamiento de la celda (0,0) se define:

$\eta = 2$

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

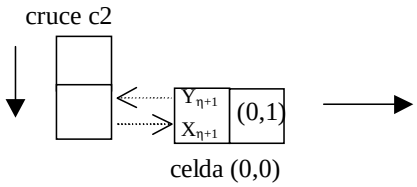


Figura 5. Vecindario/acoplamiento para la celda (0,0)

La función τ para esta celda se define de igual forma que para el resto de las celdas, a excepción de la primer regla. La definición formal de las reglas puede consultarse en [DW99].

Esta celda esta acoplada a otra celda perteneciente a un cruce (cruce c2) como puede verse en la figura 5. En las reglas de movimiento de autos definidas para esta celda se incluye el intercambio de información de la misma con la celda correspondiente del cruce c2. A través de un port (x-c-auto) el tramo se informa que hay un auto que está esperando salir del cruce para ingresar a él.

La celda (0, 9) también debe ser definida utilizando un vecindario y comportamiento distinto al resto:

$\eta = 2$
$\mathbf{N} = \{ (0,-1), (0,0) \}$

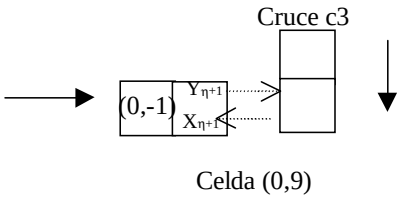


Figura 6. Vecindario/ acoplamiento de la celda (0,9)

La función τ se define como el resto del espacio de celdas, exceptuando la segunda regla. La definición formal de las reglas puede consultarse en [DW99], sección 1.1.1.

Delay = inertial.

Como puede observarse en la Figura 6, la celda (0, 9) se haya acoplada con la primer celda del cruce c3. El port x-c-haylugar indica si hay lugar en la celda del cruce, para que un auto pueda ingresar a él. Si esta celda y la celda anterior del cruce se hayan vacías, el auto puede ingresar al cruce. De esta forma, los autos que se encuentran en el cruce tienen una mayor prioridad que los autos que tratan de ingresar al mismo. En este caso se utiliza demora inercial, ya que un auto puede ingresar en un cruce únicamente si existe lugar durante un tiempo específico. De otra forma, el auto debe permanecer esperando antes de ingresar al cruce.

A continuación (figuras 7 y 8) mostramos la implementación de este tramo y de las reglas de movimiento de los autos para tramos de un carril en la herramienta N-CD++.

```
[top]
components: rA rB rC rE rF rD1 rD2 rG1 rG2 rH1 rH2 rI1 rI2 via c1 c2 c3 c4
estacion@Generator
components: genAuto1@generator genAuto2@generator genAuto3@generator genAuto4@generator
components: cont1@contador
Out : outTrans1 outTrans2

% Acoplamiento salida del cruce c2-rB
link : outAuto1@c2 inAuto@rB
link : outPuedeSalir1@c2 inPuedeEntrar@rB
link : outHayLugar@rB inHayLugar1@c2

% Acoplamiento entrada al cruce rB-c3
link : inAuto@rB inAuto0@c3
link : outHayLugar0@c3 inHayLugar@rB
link : outQuiereCruzar@rB inHayAuto0@c3
.....
.....
[rB]
type : cell
Width : 10
Height : 1
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rB(0,-1)  rB(0,0)  rB(0,1)
initialvalue : 0
localtransition : tramo-rule-1carril
in : inAuto inPuedeEntrar inhaylugar
link : inAuto inAuto@rB(0,0)
link : inPuedeEntrar inPudeEntrar@rB(0,0)
link : inHayLugar inHayLugar@rB(0,9)
out : outHayLugar outAuto outQuiereCruzar
link : outHayLugar@rB(0,0) outHayLugar
link : outAuto@rB(0,9) outAuto
link : outQuiereCruzar@rB(0,9) outQuiereCruzar
portInTransition : inAuto@rB(0,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rB(0,0) PuedeEntrarTramo
portInTransition : inHayLugar@rB(0,9) SaleACruceAuto
```

Figura 7. Implementación del tramo rB en la herramienta N-CD++

Figura 7. Implementación del tramo rB en la herramienta N-CD++

```
[PuedeEntrarTramo]
% Le indica al cruce si hay lugar para que ingrese el auto
rule : { (0,0) + send(outPuedeEntrar,(0,0)) } 10 { t }

[IngresaAutoTramo]
% Ingresa un auto al tramo (posiblemente desde un cruce)
rule : { 1 } 10 { portvalue(ThisPort)!= 0 and (0,0) = 0 }
rule : 0 10  { t }

[tramo-rule-1carril]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) = 1 and (0,1) = 0 }
% Viene de atras
rule : { 1 } 10 { not isUndefined((0,-1)) and (0,-1) = 1  and (0,0) = 0 }
% Quiere salir a cruce
rule : {(0,0)+ send(outQuiereCruzar,1)} 10 {not isundefined((0,-1)) and
       isundefined((0,1)) and  (0,0) = 1}
% Default
rule : {(0,0)} 10  { t }

[SaleACruceAuto]
% Sale hacia adelante (para la celda de salida)
rule : { 0 + send(outAuto,(0,0) ) } 0 {(0,0)=1  and portvalue(ThisPort) = 0 }
% Default
rule : {(0,0)} 10  { t }
```

Figura 8. Implementación de las reglas de movimiento de los autos para tramos de un carril en la herramienta N-CD++

Tramos con varios carriles

Así, siguiendo la idea del movimiento de lo autos se van definiendo los modelos para calles de 2, 3, 4 y más de 4 carriles. Cada uno va incrementando la cantidad de carriles y determinando cambios en las reglas de comportamiento de los autos, que cada vez deben verificar más condiciones para poder avanzar. Esto se debe a que al tener vecindarios más grandes, hay mayor posibilidad de movimientos y de conflictos por acceso a la misma celda.
Como ejemplo de tramos de más de un carril, será especificado el tramo rG2, que cuenta con cuatro carriles.

El tramo rG2= [(100,0), (100,200),4,0,0, 60] es mapeado con un Cell-DEVS de dos dimensiones con demora de transporte, cuya estructura se presenta en la figura 9.

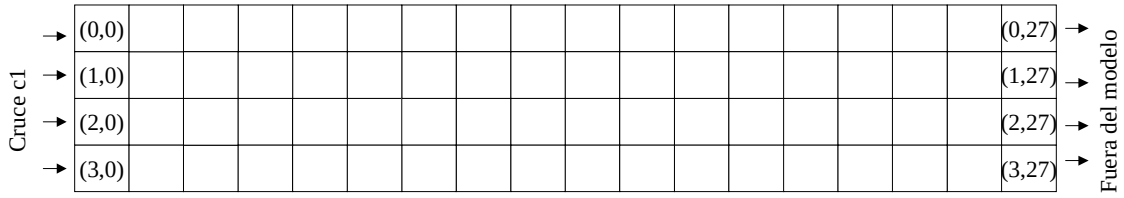


Figura 9– Tramo rG2

El número de celdas del tramo (k) se obtiene calculando la distancia entre los puntos (100,0) y (100,200) y dividiendo el resultado por el tamaño de la celda (7,5).

$$k = \left\lceil \sqrt{|x1-x2|^2 + |y1-y2|^2} \Big/ cell\_size \right\rceil = \left\lceil \sqrt{0+200^2} / 7.5 \right\rceil = 27$$

El tramo rG2 está representado por un Cell-DEVS de dos dimensiones con demora de transporte de 4 filas y 27 columnas.

Las celdas de la primera fila (carril 0) del espacio se definen como:

$$C_{0j} = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = < η, P^x, P^y>, donde

η = 8

P^x = { (X_1, binario), (X_2, binario), (X_3, binario), (X_4, binario), (X_5, binario), (X_6, binario), (X_7, binario), (X_8, binario) }

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

$\boldsymbol{X} = \boldsymbol{Y} = \{0, 1\}$

S:

$$s = \begin{cases} 1 \text{ si hay un vehículo en la celda} \\ 0 \text{ sino.} \end{cases}$$

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

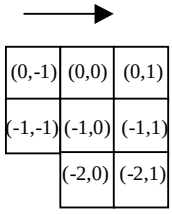


Figura 10 - Vecindario de la celda origen (carril 0)

delay = transport

d = Conversión_Demora(velocidad(max))

λ, $\boldsymbol{\delta_{int}}$ y $\boldsymbol{\delta_{ext}}$ se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.
La función **τ**: S x N → S , es la definida en [DW99] sección 1.1.1.4
Los autos que avanzan hacia estas celdas sólo lo pueden hacer desde la celda de atrás o desde el otro carril (derecho), debido a la conformación del vecindario. Luego los autos que abandonan estas celdas lo pueden hacer hacia delante o en diagonal derecha. Los autos que avanzan derecho tienen prioridad, luego los que mueven hacia izquierda y por último los que lo hacen hacia la derecha. Por esto cuando un auto quiere avanzar hacia la derecha debe verificar que no se interponga en el camino de otro vehículo.

Las celdas de la segunda fila (carril 1) del espacio se definen como:

$$C_{1j} = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = $< \eta, P^x, P^y >$, donde

$\eta = 11$

P^x = { (X_1, binario), (X_2, binario), (X_3, binario), (X_4, binario), (X_5, binario), (X_6, binario), (X_7, binario), (X_8, binario), (X_9, binario), (X_{10}, binario), (X_{11}, binario) }

P^y = { (Y_1, binario), (Y_2, binario), (Y_3, binario), (Y_4, binario), (Y_5, binario) , (Y_6, binario), (Y_7, binario), (Y_8, binario), (Y_9, binario), (Y_{10}, binario), (Y_{11}, binario) }

$\boldsymbol{X} = \boldsymbol{Y}$ = {0, 1}

S:

$$s = \begin{cases} 1 \text{ si hay un vehículo en la celda} \\ 0 \text{ sino} \end{cases}$$

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

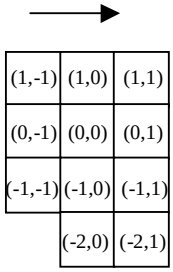


Figura 11 - Vecindario de la celda origen (carril 1)

delay = transport

d = Conversión_Demora(velocidad(max))

λ, $\boldsymbol{\delta_{int}}$ y $\boldsymbol{\delta_{ext}}$ se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

La función **τ**: S x N → S , es la definida en [DW99], sección 1.1.1.4
Los autos que avanzan hacia estas celdas y que salen de ellas lo pueden hacer utilizando los 3 movimientos posibles (derecho, diagonal izquierda y diagonal derecha).

Las celdas de la tercera fila (carril 2) del espacio se definen como:

$$C_{2j} = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = < η, P^x, P^y>, donde

η = 11

P^x = { (X_1, binario), (X_2, binario), (X_3, binario), (X_4, binario), (X_5, binario), (X_6, binario), (X_7, binario), (X_8, binario), (X_9, binario), (X_{10}, binario), (X_{11}, binario) }

P^y = { (Y_1, binario), (Y_2, binario), (Y_3, binario), (Y_4, binario), (Y_5, binario) , (Y_6, binario), (Y_7, binario), (Y_8, binario), (Y_9, binario), (Y_{10}, binario), (Y_{11}, binario) }.

$\boldsymbol{X} = \boldsymbol{Y} = \{0, 1\}$

S:

$$s = \begin{cases} 1 \text{ si hay un vehículo en la celda} \\ 0 \text{ sino.} \end{cases}$$

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

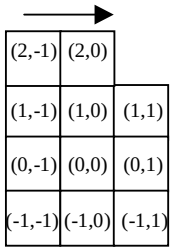


Figura 12 - Vecindario de la celda origen (carril 2)

delay = transport

d = Conversión_Demora(velocidad(max))

λ, **δ_{int}** y **δ_{ext}** se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

La función **τ**: S x N $\rightarrow$ S , es la definida en [DW99] sección 1.1.1.4
Los autos que avanzan hacia atrás estas celdas y salen de ellas lo pueden hacer utilizando los 3 movimientos posibles (derecho, diagonal izquierda y diagonal derecha).

Las celdas de la cuarta fila (carril 3) del espacio se definen como:

$$C_{3j} = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = < η, P^x, P^y>, donde

$\eta = 8$

P^x = { (X_1, binario), (X_2, binario), (X_3, binario), (X_4, binario), (X_5, binario), (X_6, binario), (X_7, binario), (X_8, binario) }

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

X = ***Y*** = {0, 1}

S:

s = { 1 si hay un vehículo en la celda
    { 0 sino.

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

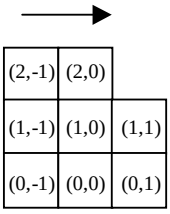


Figura 13 - Vecindario de la celda origen (carril 3)

delay = transport

d = Conversión_Demora(velocidad(max))

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

La función **τ**: S x N → S , es la definida en [DW99] sección 1.1.1.4
Los autos que avanzan hacia estas celdas sólo lo pueden hacer desde la celda de atrás o desde el otro carril (izquierdo), debido a la conformación del vecindario. Los autos primero intentan avanzar derecho, luego hacia izquierda y por último hacia derecha. Por esto, un auto avanza desde el carril izquierdo hacia la celda origen sólo cuando no puede hacer ninguno de los otros 2 movimientos (avanzar derecho y avanzar hacia la izquierda). Luego, los autos que abandonan estas celdas lo pueden hacer hacia adelante o hacia la diagonal izquierda.

El modelo acoplado correspondiente al tramo rG2 = [(100,0), (100,200), 4, 60, 0, 0] se define como:

TC4(10,60) = < Xlist, Ylist, I, *X*, *Y*, n, {t_1,...,t_n}, η, N, C, B, Z, select >

donde, T4 significa tramo de cuatro carriles de longitud 27 (celdas) cada uno, con velocidad 60 (km/h).

Ylist = { (i,0) / $0 \le i < 4$ } U { (i, 26) / $0 \le i < 4$ }

Xlist = { (i,0) / $0 \le i < 4$ } U { (i, 26) / $0 \le i < 4$ }

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

$P^x = \{<X_{\eta+1}(i,0), \text{binario}> / \ 0 \le i < 4\} \text{ U } \{<X_{\eta+1}(i,26), \text{binario}> / \ 0 \le i < 4\}$
$P^y = \{<Y_{\eta+1}(i,0), \text{binario}> / 0 \le i < 4\} \text{ U } \{<Y_{\eta+1}(i,26), \text{binario}> / \ 0 \le i < 4\}$

Estos ports serán denotados de la siguiente forma:

Port	Nombre
$X_{\eta+1}(i,0)$, $0 \le i \le 3$	x-c-hayauto
$X_{\eta+1}(i,26)$, $0 \le i \le 3$	x-c-haylugar
$Y_{\eta+1}(i,0)$, $0 \le i \le 3$	y-c-haylugar
$Y_{\eta+1}(i,26)$, $0 \le i \le 3$	y-c-hayauto

X = { 0, 1 }
Y = { 0, 1 }
n = 2
$\mathbf{t_1}$ = 4
$\mathbf{t_2}$ = 10

N y **η** han sido descriptos al definir el modelo de celda de cada carril.

C = { C_{ij} / $i \in [0, 3] \ \wedge j \in [0, 9]$ }, donde cada C_{ij} es un componente Cell-DEVS con demora y la especificación de cada celda ha sido descripta antes de introducir el modelo acoplado.

B = { (0,0), (1,0), (2,0), (3,0), (0,9), (1,9), (2,9), (3,9) }

Z se construye siguiendo la definición dada por el formalismo Cell_DEVS, instanciada con el vecindario de este espacio.

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

El comportamiento para las celdas borde es distinto al definido anteriormente.

Comportamiento y vecindad de la celda (0,0):
Para la celda (0,0) se define el siguiente comportamiento y vecindario:

η = 6
N = { (0,0), (0,1), (-1,1), (-1,0), (-2,0), (-2,1) }

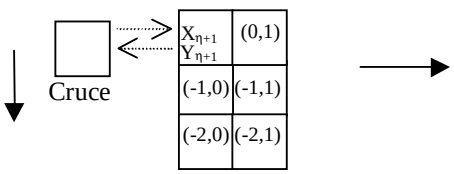


Figura 14 – Vecindario y acoplamiento de la celda (0,0)

La función τ: S x N → S , es la definida en [DW99] sección 1.1.1.4
Esta celda sólo recibe vehículos que salen del cruce, mientras que las reglas para avanzar desde ella son las mismas que las definidas para las celdas del carril 0. Los demás parámetros del modelo para esta celda no cambian.

Comportamiento y vecindad de la celda (1,0):
η = 8

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

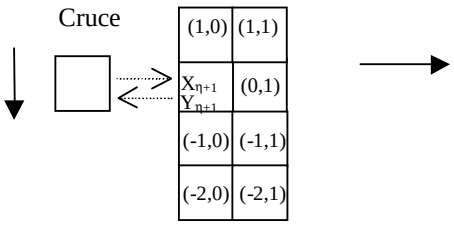


Figura 15– Vecindario y acoplamiento de la celda (1,0)

La función τ: S x N → S , es la definida en [DW99] sección 1.1.1.4
Esta celda sólo recibe vehículos que salen del cruce, mientras que las reglas para avanzar desde ella son las mismas que las definidas para las celdas del carril 1. Los demás parámetros del modelo para esta celda no cambian.

Comportamiento y vecindad de la celda (2,0):

η = 6
N = { (0,0), (0,1), (-1,1), (-1,0), (1,0), (1,1) }

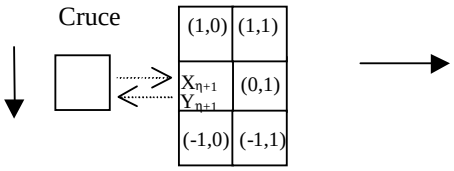


Figura 16 – Vecindario y acoplamiento de la celda (2,0)

La función τ: S x N → S , es la definida en [DW99] sección 1.1.1.4
Esta celda sólo recibe vehículos que salen del cruce, mientras que las reglas para avanzar desde ella son las mismas que las definidas para las celdas del carril 2. Los demás parámetros del modelo para esta celda no cambian.

Comportamiento y vecindad de la celda (3,0):

Para la celda (3,0) también se define un vecindario y comportamiento diferente al resto. Aquí:

$\eta = 4$
N = { (0,0), (0,1), (1,0), (1,1) }

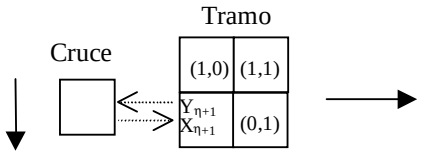


Figura 17 – Vecindario y acoplamiento de la celda (3,0)

La función τ: S x N $\rightarrow$ S , es la definida en [DW99] sección 1.1.1.4.
Esta celda sólo recibe vehículos que salen del cruce, mientras que las reglas para avanzar desde ella son las mismas que las definidas para las celdas del carril 3. Los demás parámetros del modelo para esta celda no cambian.

Comportamiento y vecindad de la celda (0,26):
Para la celda (0,26) se define el siguiente vecindario y comportamiento:

$\eta = 4$
N = { (-1,0), (-1,-1), (0,0), (0,-1) }

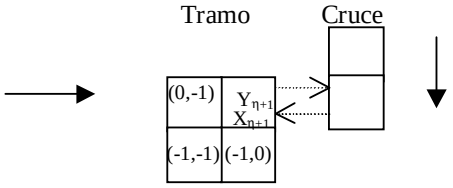


Figura 18 – Vecindario y acoplamiento de la celda (0,26)

La función τ: S x N $\rightarrow$ S , es la definida en [DW99], sección 1.1.1.4.
Las reglas para avanzar hacia esta celda son las mismas que las definidas para las celdas del carril 0; mientras que para abandonar la celda, se debe ingresar al cruce. Los demás parámetros del modelo para esta celda no cambian.

Comportamiento y vecindad de la celda (1,26)

$\eta = 6$
N = { (0,0), (0,-1), (-1,-1), (-1,0), (1,0), (1,-1) }

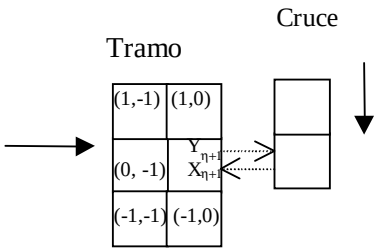


Figura 19 – Vecindario y acoplamiento de la celda (1,26)

La función τ: S x N → S , es la definida en [DW99] sección 1.1.1.4.
Las reglas para avanzar hacia esta celda son las mismas que las definidas para las celdas del carril 1, mientras que para abandonar la celda, se debe ingresar al cruce. Los demás parámetros del modelo para esta celda no cambian.

Comportamiento y vecindad de la celda (2,26):

η = 8

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

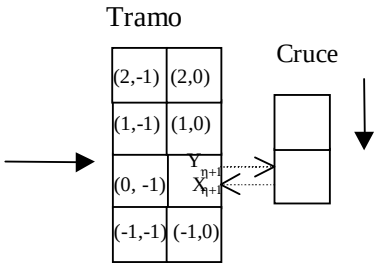


Figura 20 – Vecindario y acoplamiento de la celda (2,26)

La función τ: S x N → S , es la definida en [DW99], sección 1.1.1.4.
Las reglas para avanzar hacia esta celda son las mismas que las definidas para las celdas del carril 2, mientras que para abandonar la celda, se debe ingresar al cruce. Los demás parámetros del modelo para esta celda no cambian.

Comportamiento y vecindad de la celda (3,26):

η = 6

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

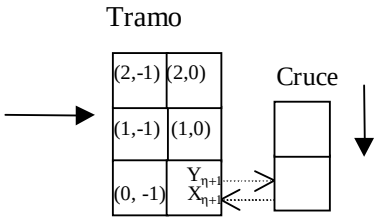


Figura 21– Vecindario y acoplamiento de la celda (3,26)

La función τ: S x N $\rightarrow$ S , es la definida en [DW99] sección 1.1.1.4.
Las reglas para avanzar hacia esta celda son las mismas que las definidas para las celdas del carril 3, mientras que para abandonar la celda, se debe ingresar al cruce. Los demás parámetros del modelo para esta celda no cambian.

A continuación mostramos una parte de la implementación del tramo rG2 en la herramienta N-CD++. La definición completa se encuentra descripta en el apéndice B.

```
[top]
....
% Acoplamiento salida del cruce c1-rG2
link : outAuto0@c1 inAuto3@rG2
link : outAuto1@c1 inAuto2@rG2
link : outAuto2@c1 inAuto1@rG2
link : outAuto3@c1 inAuto0@rG2
link : outPuedeSalir0@c1 inPuedeEntrar3@rG2
link : outPuedeSalir1@c1 inPuedeEntrar2@rG2
link : outPuedeSalir2@c1 inPuedeEntrar1@rG2
link : outPuedeSalir3@c1 inPuedeEntrar0@rG2
link : outPuedeEntrar3@rG2 inHayLugar0@c1
link : outPuedeEntrar2@rG2 inHayLugar1@c1
link : outPuedeEntrar1@rG2 inHayLugar2@c1
link : outPuedeEntrar0@rG2 inHayLugar3@c1
......
[rG2]
type : cell
Width : 27
Height : 4
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rG2(1,-1)  rG2(1,0)  rG2(1,1)
neighbors : rG2(0,-1)  rG2(0,0)  rG2(0,1)
neighbors : rG2(-1,-1) rG2(-1,0) rG2(-1,1)
initialvalue : 0
localtransition : tramo-rule-4carril
in : inAuto0 inAuto1 inAuto2 inAuto3
in : inPuedeEntrar0 inPuedeEntrar1 inPuedeEntrar2 inPuedeEntrar3
in : inHayLugar0 inHayLugar1 inHayLugar2 inHayLugar3
link : inAuto0 inAuto@rG2(0,0)
link : inAuto1 inAuto@rG2(1,0)
link : inAuto2 inAuto@rG2(2,0)
link : inAuto3 inAuto@rG2(3,0)
link : inPuedeEntrar0 inPuedeEntrar@rG2(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rG2(1,0)
link : inPuedeEntrar2 inPuedeEntrar@rG2(2,0)
link : inPuedeEntrar3 inPuedeEntrar@rG2(3,0)
link : inHayLugar0 inHayLugar@rG2(0,26)
link : inHayLugar1 inHayLugar@rG2(1,26)
link : inHayLugar2 inHayLugar@rG2(2,26)
link : inHayLugar3 inHayLugar@rG2(3,26)
out : outPuedeEntrar0 outPuedeEntrar1 outPuedeEntrar2 outPuedeEntrar3
out : outAuto0 outAuto1 outAuto2 outAuto3
out : outQuiereCruzar0 outQuiereCruzar1  outQuiereCruzar2 outQuiereCruzar3
link : outPuedeEntrar@rG2(0,0) outPuedeEntrar0
link : outPuedeEntrar@rG2(1,0) outPuedeEntrar1
link : outPuedeEntrar@rG2(2,0) outPuedeEntrar2
link : outPuedeEntrar@rG2(3,0) outPuedeEntrar3
link : outAuto@rG2(0,26) outAuto0
link : outAuto@rG2(1,26) outAuto1
link : outAuto@rG2(2,26) outAuto2
link : outAuto@rG2(3,26) outAuto3
link : outQuiereCruzar@rG2(0,26) outQuiereCruzar0
link : outQuiereCruzar@rG2(1,26) outQuiereCruzar1
link : outQuiereCruzar@rG2(2,26) outQuiereCruzar2
link : outQuiereCruzar@rG2(3,26) outQuiereCruzar3
zone : SaleFueraModelo { (0,26), (1,26), (2,26), (3,26) }
portInTransition : inAuto@rG2(0,0) IngresaAutoTramo
portInTransition : inAuto@rG2(1,0) IngresaAutoTramo
portInTransition : inAuto@rG2(2,0) IngresaAutoTramo
portInTransition : inAuto@rG2(3,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rG2(0,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG2(1,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG2(2,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG2(3,0) PuedeEntrarTramo
portInTransition : inHayLugar@rG2(0,26) SaleACruceAuto
portInTransition : inHayLugar@rG2(1,26) SaleACruceAuto
portInTransition : inHayLugar@rG2(2,26) SaleACruceAuto
portInTransition : inHayLugar@rG2(3,26) SaleACruceAuto
```

Figura 22. Implementación del tramo rG2 en la herramienta N-CD++

Cruces

Los cruces mostrados en la figura 1 se definen del siguiente modo, utilizando la definición dada en la sección 1.1.2. de [DW99]

c1 = ((100,200); 30)
c2 = ((0,130); 30)
c3 =((0,200); 30)
c4 = ((0,300); 30)

Por ejemplo, el cruce c1 se define como un modelo Cell-DEVS de una dimensión con demora de transporte (puede ser visto en la figura 23). Este cruce tiene 4 tramos de ingreso que son: rG1, rH2, rD1 y rI2.

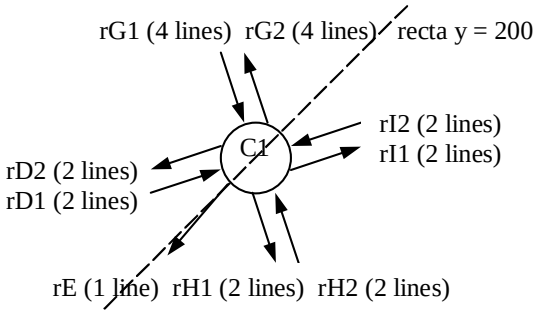


Figura 23. Cruce c1

Las celdas del cruce se definen del mismo modo que las celdas de un modelo de un solo carril, a excepción de las reglas que definen el comportamiento de las celdas. En este caso, cada celda se haya conectada con un tramo, el cuál tiene influencia sobre su comportamiento.

Como se observa en la Figura 23, el cruce c1 está conectado a 9 tramos (rG1, rG2, rI1, rI2, rE, rH1, rH2, rD1 y rD2), de los cuales algunos son de ingreso al cruce y otros para salida de vehículos. Así se pueden construir los siguientes conjuntos con tramos de entrada y salida, respectivamente:

T_{in} = { rG1, rI2, rH2, rD1 }

T_{out} = { rG2, rI1, rE, rH1, rD2 }

Como el cruce tiene una celda por cada celda de cada tramo que se acopla, el número de celdas del cruce se calcula como:

$$k = \sum_{t \in (T_{in} \cup T_{iout})} n = 4+4+2+2+2+2+1+2+2 = 21$$

Para establecer la posición en que cada tramo se acopla al cruce, se define un ordenamiento según el ángulo de inclinación de los mismos respecto a la recta y = 200 con c1 = (100,200). Así, a medida que los tramos tengan mayor ángulo les corresponderán las posiciones mayores dentro del cruce. Logrando que los tramos cercanos ocupen las celdas adyacentes dentro del cruce.

Entonces, las primeras celdas del cruce se acoplan con rG2, las siguientes con rG1, y así siguiendo. Esto es:

{(0, 0), (0,1), (0,2), (0,3)} celdas de acoplamiento para rG2
{(0, 4), (0,5), (0,6), (0,7)} celdas de acoplamiento para rG1
{(0, 8), (0,9) } celdas de acoplamiento para rD2
{(0, 10), (0,11) } celdas de acoplamiento para rD1
{(0, 12) } celdas de acoplamiento para rE
{(0, 13), (0,14) } celdas de acoplamiento para rH1
{(0, 15), (0,16) } celdas de acoplamiento para rH2
{(0, 17), (0,18) } celdas de acoplamiento para rI1
{(0, 19), (0,20) } celdas de acoplamiento para rI2

Por lo tanto, a partir de ellos se definen los siguientes conjuntos que representan las posiciones del cruce que son entradas y salidas, respectivamente:

In = { 4, 5, 6, 7, 10, 11, 15, 16, 19, 20}

Out = {0, 1, 2, 3, 8, 9, 12, 13, 14, 17, 18}

El modelo acoplado correspondiente al cruce se define como:

Cruce(21, In, Out) = < Xlist, Ylist, I, *X*, *Y*, n, $\{t_1,...,t_n\}$, η, N, C, B, Z, select >

En este caso, Cruce(21, In, Out) define un cruce de 21 celdas, donde las posiciones del conjunto In representan entradas al cruce y Out salidas.
Este modelo representa el movimiento de vehículos en las intersecciones de tramos. Cada celda del cruce se halla conectada con un tramo, por lo tanto la interface del modelo incluye a todas las celdas.
Los ports de entrada y salida se utilizan para informar el estado de cada celda a los tramos y viceversa.

La definición de cada conjunto en el modelo es la siguiente:

Ylist = **Xlist** = { (0,i) / $0 \leq i < 21$}

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,i), binario> / $0 \leq i < 21$ }
P^y = {<$Y_{\eta+1}$(0,i), binario> / $0 \leq i < 21$ }

Los ports serán llamados:

Port	Nombre
$X_{\eta+1}(0,i)$, $i \in$ In	x-t-auto
$X_{\eta+1}(0,i)$, $i \in$ Out	x-t-HayLugar
$Y_{\eta+1}(0,i)$, $i \in$ In	y-t- HayLugar
$Y_{\eta+1}(0,i)$, $i \in$ Out	y-t- auto

X = ***Y*** = { 0, 1 }

n = 1

$\mathbf{t_1}$ = 21

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

B = {∅}

τ: S x N → S,

La función **τ,** que define el comportamiento de las celdas será diferente dependiendo de si la celda es una celda de entrada o de salida al cruce. Estas reglas fueron definidas en la sección 1.1.2 de [DW99].

Acoplamiento para el cruce c1 y los tramos correspondientes

I_{c1} = { rG1, rG2, rI1, rI2, rE, rH1, rH2, rD1 y rD2 } = T_{in} U T_{out}

Luego un tramo t tiene definidas las influencias sobre los modelos de los dos cruces que tienen como extremo, por ejemplo:

$I_{rG1} = \{M_{c1}\}$
$I_{rG2} = \{M_{c1}\}$
$I_{rE} = \{M_{c3}\} \cup \{M_{c1}\}$
$I_{rD1} = \{M_{c2}\} \cup \{M_{c1}\}$
$I_{rD2} = \{M_{c2}\} \cup \{M_{c1}\}$

Para completar la definición del acoplamiento entre cruces y tramos hay que establecer a través de qué ports se comunican.

Acoplamiento con tramo rG2 (rG2 tiene 27 celdas):
La definición de Z para rG2 y c1 es:

$$Z_{c1\_rG2} : Y_{\eta+1}(0,j)_{c1} \rightarrow X_{\eta+1}(3\text{-}j, 0)_{rG2}, \forall (j \in N, j \in [0, 3])$$
$$Z_{rG2\_c1} : Y_{\eta+1}(3\text{-}j, 0)_{rG2} \rightarrow X_{\eta+1}(0,j)_{c1}, \forall (j \in N, j \in [0, 3])$$

Acoplamiento con tramo rG1 (rG1 tiene 27 celdas):
La definición de Z para rG1 y c1 es:

$$Z_{rG1\_c1} : Y_{\eta+1}(j, 26)_{rG1} \rightarrow X_{\eta+1}(0,4+j)_{c1}, \forall (j \in N, j \in [0, 3])$$
$$Z_{c1\_rG1} : Y_{\eta+1}(0,4+j)_{c1} \rightarrow X_{\eta+1}(j, 26)_{rG1}, \forall (j \in N, j \in [0, 3])$$

Acoplamiento con tramo rD2 (rD2 tiene 17 celdas):
La definición de Z para rD2 y c1 es:

$$Z_{c1\_rD2} : Y_{\eta+1}(0,j+8)_{c1} \rightarrow X_{\eta+1}(1\text{-}j, 0)_{rD2}, \forall (j \in N, j \in [0, 1])$$
$$Z_{rD2\_c1} : Y_{\eta+1}(1\text{-}j, 0)_{rD2} \rightarrow X_{\eta+1}(0,j+8)_{c1}, \forall (j \in N, j \in [0, 1])$$

Acoplamiento con tramo rD1(rD1 tiene 17 celdas):
La definición de Z para rD1 y c1 es:

$$Z_{rD1\_c1} : Y_{\eta+1}(j, 16)_{rD1} \rightarrow X_{\eta+1}(0,10+j)_{c1}, \forall (j \in N, j \in [0, 1])$$
$$Z_{c1\_rD1} : Y_{\eta+1}(0,10+j)_{c1} \rightarrow X_{\eta+1}(j, 16)_{rD1}, \forall (j \in N, j \in [0, 1])$$

Acoplamiento con tramo rE (rE tiene 14 celdas):
La definición de Z para rE y c1 es:

$$Z_{c1\_rE} : Y_{\eta+1}(0,12)_{c1} \rightarrow X_{\eta+1}(0, 0)_{rE}$$
$$Z_{rE\_c1} : Y_{\eta+1}(0, 0)_{rE} \rightarrow X_{\eta+1}(0,12)_{c1}$$

Acoplamiento con tramo rH1 (rH1 tiene 14 celdas):
La definición de Z para rH1 y c1 es:

$$Z_{c1\_rH1} : Y_{\eta+1}(0,j +13)_{c1} \rightarrow X_{\eta+1}(1-j, 0)_{rH1}, \forall (j \in N, j \in [0, 1])$$
$$Z_{rH1\_c1} : Y_{\eta+1}(1-j, 0)_{rH1} \rightarrow X_{\eta+1}(0,j+13)_{c1}, \forall (j \in N, j \in [0, 1])$$

Acoplamiento con tramo rH2 (rH2 tiene 14 celdas):
La definición de Z para rH2 y c1 es:

$$Z_{rH2\_c1} : Y_{\eta+1}(j, 13)_{rH2} \rightarrow X_{\eta+1}(0,15+j)_{c1}, \forall (j \in N, j \in [0, 1])$$
$$Z_{c1\_rH2} : Y_{\eta+1}(0,15+j)_{c1} \rightarrow X_{\eta+1}(j, 13)_{rH2}, \forall (j \in N, j \in [0, 1])$$

Acoplamiento con tramo rI1(rI1 tiene 18 celdas):
La definición de Z para rI1 y c1 es:

$$Z_{c1\_rI1} : Y_{\eta+1}(0,j +17)_{c1} \rightarrow X_{\eta+1}(1-j, 0)_{rI1}, \forall (j \in N, j \in [0, 1])$$
$$Z_{rI1\_c1} : Y_{\eta+1}(1-j, 0)_{rI1} \rightarrow X_{\eta+1}(0,j+17)_{c1}, \forall (j \in N, j \in [0, 1])$$

Acoplamiento con tramo rI2 (rI2 tiene 18 celdas):
La definición de Z para rI2 y c1 es:

$$Z_{rI2\_c1} : Y_{\eta+1}(j, 17)_{rI2} \rightarrow X_{\eta+1}(0,19+j)_{c1}, \forall (j \in N, j \in [0, 1])$$
$$Z_{c1\_rI2} : Y_{\eta+1}(0,19+j)_{c1} \rightarrow X_{\eta+1}(j, 17)_{rI2}, \forall (j \in N, j \in [0, 1])$$

En las figuras 24 y 25 mostramos la implementación del cruce c1 en la herramienta N-CD++.

```
[top]
....
% Acoplamiento entrada al cruce rG1-c1
link : outAuto0@rG1 inAuto4@c1
link : outAuto1@rG1 inAuto5@c1
link : outAuto2@rG1 inAuto6@c1
link : outAuto3@rG1 inAuto7@c1
link : outHayLugar4@c1 inHayLugar0@rG1
link : outHayLugar5@c1 inHayLugar1@rG1
link : outHayLugar6@c1 inHayLugar2@rG1
link : outHayLugar7@c1 inHayLugar3@rG1
link : outQuiereCruzar0@rG1 inHayAuto4@c1
link : outQuiereCruzar1@rG1 inHayAuto5@c1
link : outQuiereCruzar2@rG1 inHayAuto6@c1
link : outQuiereCruzar3@rG1 inHayAuto7@c1

% Acoplamiento entrada al cruce rD1-c1
link : outAuto0@rD1 inAuto10@c1
link : outAuto1@rD1 inAuto11@c1
link : outHayLugar10@c1 inHayLugar0@rD1
link : outHayLugar11@c1 inHayLugar1@rD1
link : outQuiereCruzar0@rD1 inHayAuto10@c1
link : outQuiereCruzar1@rD1 inHayAuto11@c1

% Acoplamiento entrada al cruce rH2-c1
link : outAuto0@rH2 inAuto15@c1
link : outAuto1@rH2 inAuto16@c1
link : outHayLugar15@c1 inHayLugar0@rH2
link : outHayLugar16@c1 inHayLugar1@rH2
link : outQuiereCruzar0@rH2 inHayAuto15@c1
link : outQuiereCruzar1@rH2 inHayAuto16@c1

% Acoplamiento entrada al cruce rI2-c1
link : outAuto0@rI2 inAuto19@c1
link : outAuto1@rI2 inAuto20@c1
link : outHayLugar19@c1 inHayLugar0@rI2
link : outHayLugar20@c1 inHayLugar1@rI2
link : outQuiereCruzar0@rI2 inHayAuto19@c1
link : outQuiereCruzar1@rI2 inHayAuto20@c1

% Acoplamiento salida del cruce c1-rD2
link : outAuto8@c1 inAuto0@rD2
link : outAuto9@c1 inAuto1@rD2
link : outPuedeSalir8@c1 inPuedeEntrar0@rD2
link : outPuedeSalir9@c1 inPuedeEntrar1@rD2
link : outPuedeEntrar0@rD2 inHayLugar8@c1
link : outPuedeEntrar1@rD2 inHayLugar9@c1

% Acoplamiento salida del cruce c1-rE
link : outAuto12@c1 inAuto@rE
link : outPuedeSalir12@c1 inPuedeEntrar@rE
link : outPuedeEntrar@rE inHayLugar12@c1

% Acoplamiento salida del cruce c1-rG2
link : outAuto0@c1 inAuto3@rG2
link : outAuto1@c1 inAuto2@rG2
link : outAuto2@c1 inAuto1@rG2
link : outAuto3@c1 inAuto0@rG2
link : outPuedeSalir0@c1 inPuedeEntrar3@rG2
link : outPuedeSalir1@c1 inPuedeEntrar2@rG2
link : outPuedeSalir2@c1 inPuedeEntrar1@rG2
link : outPuedeSalir3@c1 inPuedeEntrar0@rG2
link : outPuedeEntrar3@rG2 inHayLugar0@c1
link : outPuedeEntrar2@rG2 inHayLugar1@c1
link : outPuedeEntrar1@rG2 inHayLugar2@c1
link : outPuedeEntrar0@rG2 inHayLugar3@c1

% Acoplamiento salida del cruce c1-rH1
link : outAuto14@c1 inAuto0@rH1
link : outAuto13@c1 inAuto1@rH1
link : outPuedeSalir14@c1 inPuedeEntrar0@rH1
link : outPuedeSalir13@c1 inPuedeEntrar1@rH1
.....
.....
```

Figura 24. Implementación del cruce c1 en la herramienta N-CD++ (acoplamiento con otros modelos)

```
[c1]
type : cell
Width : 21
Height : 1
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : c1(0,-1) c1(0,0) c1(0,1)
initialvalue : 0
localtransition : autoCruce-entrada
in : inHayLugar0 inHayLugar1 inHayLugar2 inHayLugar3 inAuto4 inHayAuto4 inAuto5
inHayAuto5
in : inAuto6 inHayAuto6 inAuto7 inHayAuto7 inHayLugar8 inHayLugar9
in : inAuto10 inHayAuto10 inAuto11 inHayAuto11 inHayLugar12 inHayLugar13 inHayLugar14
in : inAuto15 inHayAuto15 inAuto16 inHayAuto16 inHayLugar17 inHayLugar18
in : inAuto19 inHayAuto19 inAuto20 inHayAuto20
link : inHayLugar0 inHayLugar@c1(0,0)
link : inHayLugar1 inHayLugar@c1(0,1)
link : inHayLugar2 inHayLugar@c1(0,2)
link : inHayLugar3 inHayLugar@c1(0,3)
link : inAuto4 inAuto@c1(0,4)
link : inHayAuto4 inHayAuto@c1(0,4)
link : inAuto5 inAuto@c1(0,5)
link : inHayAuto5 inHayAuto@c1(0,5)
link : inAuto6 inAuto@c1(0,6)
link : inHayAuto6 inHayAuto@c1(0,6)
link : inAuto7 inAuto@c1(0,7)
link : inHayAuto7 inHayAuto@c1(0,7)
link : inHayLugar8 inHayLugar@c1(0,8)
link : inHayLugar9 inHayLugar@c1(0,9)
link : inAuto10 inAuto@c1(0,10)
link : inHayAuto10 inHayAuto@c1(0,10)
link : inAuto11 inAuto@c1(0,11)
link : inHayAuto11 inHayAuto@c1(0,11)
link : inHayLugar12 inHayLugar@c1(0,12)
link : inHayLugar13 inHayLugar@c1(0,13)
link : inHayLugar14 inHayLugar@c1(0,14)
link : inAuto15 inAuto@c1(0,15)
link : inHayAuto15 inHayAuto@c1(0,15)
link : inAuto16 inAuto@c1(0,16)
link : inHayAuto16 inHayAuto@c1(0,16)
link : inHayLugar17 inHayLugar@c1(0,17)
link : inHayLugar18 inHayLugar@c1(0,18)
link : inAuto19 inAuto@c1(0,19)
link : inHayAuto19 inHayAuto@c1(0,19)
link : inAuto20 inAuto@c1(0,20)
link : inHayAuto20 inHayAuto@c1(0,20)
out : outAuto0 outPuedeSalir0 outAuto1 outPuedeSalir1 outAuto2 outPuedeSalir2
out : outAuto3 outPuedeSalir3 outHayLugar4 outHayLugar5 outHayLugar6 outHayLugar7
out : outAuto8 outPuedeSalir8  outAuto9 outPuedeSalir9 outHayLugar10 outHayLugar11
out : outAuto12 outPuedeSalir12 outAuto13 outPuedeSalir13 outAuto14 outPuedeSalir14
out : outHayLugar15 outHayLugar16 outAuto17 outPuedeSalir17 outAuto18 outPuedeSalir18
out : outHayLugar19 outHayLugar20
link : outAuto@c1(0,0) outAuto0
link : outPuedeSalir@c1(0,0) outPuedeSalir0
link : outAuto@c1(0,1) outAuto1
link : outPuedeSalir@c1(0,1) outPuedeSalir1
link : outAuto@c1(0,2) outAuto2
link : outPuedeSalir@c1(0,2) outPuedeSalir2
link : outAuto@c1(0,3) outAuto3
link : outPuedeSalir@c1(0,3) outPuedeSalir3
link : outHayLugar@c1(0,4) outHayLugar4
link : outHayLugar@c1(0,5) outHayLugar5
link : outHayLugar@c1(0,6) outHayLugar6
link : outHayLugar@c1(0,7) outHayLugar7
link : outAuto@c1(0,8) outAuto8
link : outPuedeSalir@c1(0,8) outPuedeSalir8
link : outAuto@c1(0,9) outAuto9
link : outPuedeSalir@c1(0,9) outPuedeSalir9
link : outHayLugar@c1(0,10) outHayLugar10
```

Figura 25. Implementación del cruce c1 en la herramienta N-CD++ (definición)

Semáforos

La presencia de semáforos se representa utilizando modelos adicionales al cruce y tramos afectados. Se define un modelo DEVS (Semáforo) para cada calle de la intersección, que informa el color del semáforo a las celdas del tramo. Luego, por cada cruce se construye un modelo DEVS (Sincronizador) encargado de avisar a cada semáforo cuándo le corresponde la luz verde, es decir sincroniza todos los semáforos del cruce. Estos modelos se grafican en la Figura 26.
Si en el cruce c1 hay un semáforo se indica agregando este cruce al conjunto *CrucesSemáforos*. Este cruce tiene 4 tramos de ingreso que son: rG1, rH2, rD1 y rI2 (T_{in}), lo que significa que se generarán 4 modelos DEVS de semáforos y un modelo DEVS sincronizador.

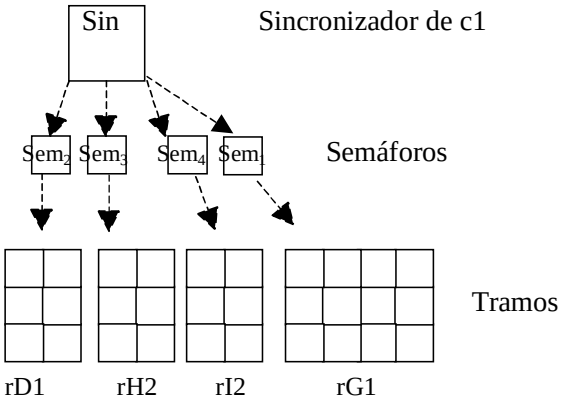


Figura 26– Modelos para representar semáforos en el cruce c1

El comportamiento del sincronizador se representa de la siguiente manera [DW99]:

λ(s)
{
send 0 *to* y-se-luz$_{SemVerde}$ /* envía luz verde al semáforo correspondiente */
send 1 (rojo) *to* y-se-luz$_j$ ($\forall$j $0 \le j < 4 \wedge j \neq SemVerde$) /* envía luz roja al */
/* resto de los semáforos */
}

δ_{int}(s,e)
{
case phase

activa:
SemVerde = (SemVerde + 1) mod 4
phase = activa
$\sigma = t_{verde}$
pasiva:
/* Nunca ocurre pues no hay evento planificado */
end case
}

El comportamiento modelado a través de estas funciones corresponde a enviar la luz verde a los semáforos conectados a este sincronizador en forma alternada. Cada semáforo recibirá la luz verde durante el tiempo t_{verde}, luego del cual volverá a rojo hasta que le vuelva a tocar su turno. En cada momento un único semáforo de todos los conectados al sincronizador recibe la luz verde; mientras tanto los demás están en rojo.

El comportamiento de cada semáforo se describe con las siguientes funciones (en particular se describe el que se acopla a rD1):

δ_{ext}(s,e,x)
{
cuando recibe x en el port sincro
case phase
activa:
/* no debería ocurrir */
pasiva:
ColorSem = x
phase = activa
$\sigma = 0$
end case
}

λ(s)
{
send ColorSem *to* y-t-luz$_i$ ($\forall i \in N, 0 \leq i < 2$)
}

δ_{int}(s,e)
{
case phase
activa:
phase = pasiva
$\sigma = \infty$
pasiva:
/* Nunca ocurre */
end case
}

Cada semáforo regula el ingreso de los vehículos de un *sólo tramo* al cruce. El comportamiento modelado por el semáforo consiste en recibir la luz (roja o verde) que le envía el sincronizador e informar la misma a las celdas del tramo sobre las que tiene influencia. Cuando el semáforo recibe el verde del sincronizador, habilita a los vehículos del tramo para que ingresen al cruce. Las celdas del cruce no necesitan conocer el estado del semáforo, pues con el estado de las celdas del tramo les alcanza para determinar si el auto avanza o no.

Para cada tramo de T_{in}, se debe reflejar en sus reglas de comportamiento la presencia del semáforo. Para estos tramos se modifica el comportamiento de las celdas de la última columna del espacio, que ahora tendrán un port externo adicional que indicará el color de la luz. Por ejemplo, para rD1 se agregan los ports:

Nota: rD1 y rD2 tienen

$$\left\lceil \sqrt{100^2 + 70^2} \Big/ 7.5 \right\rceil = 17$$

celdas.

{<$X_{\eta+2}$(0, 16), binario>, <$X_{\eta+2}$(1, 16), binario> } denotados como x-se-luz.

Además se modifica el comportamiento de las celdas borde de la columna 16 del espacio, pues deben chequear la luz del semáforo. Se agrega una nueva regla que representa que un vehículo ingresa al cruce desde el tramo, si hay lugar y si el semáforo está en verde (portvalue(x-se-luz)=0). La definición formal de las reglas que definen el comportamiento de las celdas puede consultarse en [DW99].

Esta nueva regla representa que un vehículo ingresa al cruce desde el tramo, si hay lugar y si el semáforo está en verde (portvalue(x-se-luz) = 0).

El sincronizador influye sobre el comportamiento de los modelos Semáforos

$$Isin = \{ Msem_1, Msem_2 , Msem_3, Msem_4\}$$

Cada semáforo influye sobre el comportamiento del modelo del tramo:

$$Isem_1 = \{ M_{rG1} \}; Isem_2 = \{ M_{rD1} \}; Isem_3 = \{ M_{rH2} \}; Isem_4 = \{ M_{rI2} \}$$

Luego el acoplamiento se define como:

$Zsin\,sem_i$: (y-se-luz$_i$)sin $\rightarrow$ (x-si-luz)sem$_i$, $\forall$ (i $\in$ N, $0 \leq i < 4$)

Para el tramo (ejemplificado en rD1) se define Z como:

$Zsem_{2\ rD1}$: (y-t-luz$_i$)sem$_2$ $\rightarrow$ $X_{\eta+2}(i, 16)_{rD1}$, $\forall$ (i $\in$ N, $0 \leq i < 2$)

Ferrocarril

En la figura 27 se observan las vías del tren, que atraviesan los tramos rI1 y rI2. De ahí, que el conjunto VíasTren contenga los siguientes elementos:
(rI1, 90, 1), (rI2, 10, 1)

Para cada elemento del conjunto RedDeVías se definen 2 modelos, el primero es un DEVS que representa la estación de donde parten los trenes y el segundo es un Cell-DEVS de una dimensión donde cada celda controla un paso a nivel de la vía. La estación lo único que hace es simular la partida de trenes cada cierto tiempo, de acuerdo a su frecuencia. El Cell-DEVS VíasTren representa el movimiento del tren que avanza con la velocidad modelada con demora de transporte. Estos modelos se grafican en la figura 27.

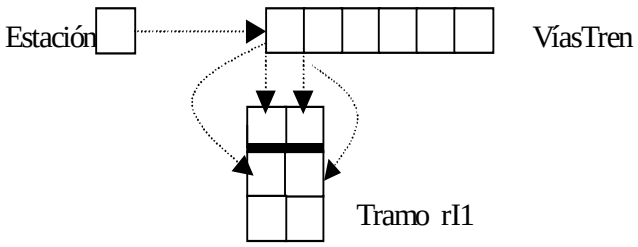


Figura 27 – Modelos para representar los trenes

Comportamiento de la estación:

```
λ(s)
{
 send tren_partiendo to y-vt-tren  /* informa de la partida del tren al modelo VíasTren */
}

δint(s,e)
{
 case tren_partiendo
        1:
                tren_partiendo = 0  /* planifica evento para avisar que la estación */
                phase = activa          /* está vacía */
                σ = ξ
        0:
                tren_partiendo = 1  /* planifica evento para el próximo tren */
                phase = activa
                σ = frec
 end case
}
```

Este modelo representa la partida de trenes de acuerdo a una frecuencia. La estación tiene un único port de salida (y-vt-tren) que lo acopla con un modelo VíasTren. Cuando su estado es 1 indica que está partiendo un tren desde la estación, por lo que comienza a circular por las vías correspondientes. Cuando su estado es 0, indica que no hay ningún tren listo para salir de la estación. Cada cierto tiempo, dado por el parámetro frec, se modela la partida de un nuevo tren, planificando el evento correspondiente. La constante ξ modela el tiempo en que tarda el tren hasta salir de la estación, es decir una vez transcurrido ésta vuelve a estar vacía (tren_partiendo = 0).

Las Vías se representan como un espacio Cell_DEVS de 1 dimensión con las siguientes características:

$\boldsymbol{X} = \boldsymbol{Y} = \{0, 1\}$

S:

$$s = \begin{cases} 1 \text{ si está el tren} \\ 0 \text{ sino} \end{cases}$$

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

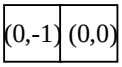


Figura 28 - Vecindario de la celda origen

delay = transport

d = d_{tren} ($d_{tren} \in N$)

El estado de una celda representa la presencia (s = 1) o ausencia (s = 0) del tren. Como sólo se permite que el tren avance hacia la posición de adelante, el vecindario necesario para establecer el nuevo estado de una celda está constituido por ella misma y la celda anterior.

Las demoras de transporte se utilizan para representar la velocidad del tren, es decir su llegada al próximo paso a nivel se demora el tiempo dado por la constante d_{tren}.

El movimiento de los trenes (especificado con la función τ definida en la sección 1.2.2. de [DW99]) queda definido por 3 reglas. La primera representa que la celda se vacía luego que el tren haya pasado. La segunda representa el avance del tren desde el paso nivel anterior hacia la celda de origen. Por último la tercer regla indica que el estado de la celda de origen no es modificado en cualquier otro caso.

El modelo acoplado correspondiente a las vías se define como:

$$VíasTren(k, Out) = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, \eta, N, C, B, Z, select >$$

donde,

Ylist = $\{(0,i) / i \in N \wedge 0 \leq i < k \}$

Xlist = { (0,0) }

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

$P^x = \{<X_{\eta+1}(0,0), binario>\}$
$P^y = \{<Y_{\eta+j}(0,i), binario> / i \in N \wedge 0 \leq i < k \wedge (i, \#p) \in Out \wedge 1 \leq j \leq \#p \}$

Estos ports serán denotados de la siguiente forma:

Port	Nombre
$X_{\eta+1}$	x-e-tren
$Y_{\eta+j}$	y-t-tren

X = ***Y*** = { 0, 1 }

n = 1; $\mathbf{t_1}$ = k ; $\boldsymbol{\eta}$ = 2

B = { (0,0) }

VíasTren(k, Out) modela un trazado de vías donde:
k representa la cantidad de pasos a nivel que tienen las vías del tren y se obtiene contando los elementos de conjunto:

Vías = { (t, ℓ, seq) / t $\in$ Tramos $\wedge$ $\ell \in$ N $\wedge$ seq $\in$ N}.

Es decir, k = #(Vías); y

el conjunto Out contiene la cantidad de ports para el acoplamiento con los tramos que tiene cada celda, y se construye como:

Out = { (i, 2*n) / i $\in$ N $\wedge$ $0 \leq i < k \wedge$ ((c1, c2, n, a, dir, max), ℓ, i) $\in$ Vías }

Cada celda tendrá tantos ports de salida externos como 2 veces la cantidad de carriles del tramo al que se acopla; pues la presencia del tren debe ser informada a 2 celdas de cada carril.

La especificación de este modelo representa el movimiento del tren, donde cada celda estará acoplada a un tramo e informará de la presencia del tren por las vías, prohibiendo la circulación de vehículos durante un lapso de tiempo.

El acoplamiento externo de este modelo se define a través de la celda (0,0) con una estación, es decir cuando el estado de este port externo de entrada es 1 indica el avance del tren hacia el modelo. Además cada celda tiene ports de salida externos hacia 2 celdas de cada carril del tramo donde se encuentra el paso a nivel correspondiente para informar la presencia del tren (s = 1). Uno de esos ports se utiliza para la celda atravesada por las vías, el otro port es para la celda que está a continuación de ellas. Estas últimas deben conocer si el tren pasó o no para saber si los vehículos se quedaron o no en la celda anterior.

El comportamiento para las celdas borde es distinto al definido anteriormente. Para la celda (0,0) se define el siguiente vecindario y comportamiento:

η = 1

N = { (0,0) }

La función τ corresponde a la definida en [DW99].

La celda (0,0) presenta comportamiento distinto porque recibe los trenes desde la estación, en vez de su vecino anterior. Por lo tanto las reglas son las mismas que las definidas para el resto de las celdas pero tienen un port de entrada que representa el estado de la estación. El resto de los parámetros de esta celda no cambian.

La tupla (rI1, 90, 1), representa que las vías atraviesan el tramo rI1. Las celdas afectadas por la barrera son aquellas de la columna :

$$col = \lceil 90 / 7.5 \rceil = 12$$

y se obtienen las celdas:

$$C_{pn} = \{ (0,12), (1,12) \} \cup \{ (0,13), (1,13) \}$$

Es decir, las celdas afectadas por el tren son aquellas que se encuentran justo antes y justo después de las vías, porque las primeras no avanzan si el tren está cerca y en ese caso las otras no reciben autos. Además el único movimiento permitido para los autos que atraviesan las vías es recto. Estas celdas se pueden ver en la Figura 29.

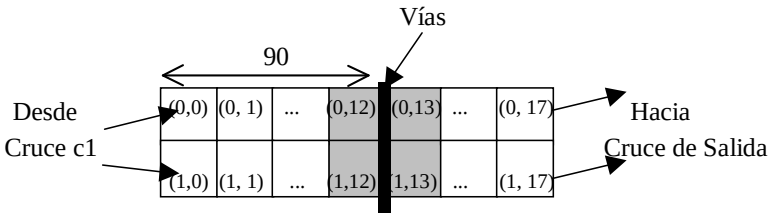


Figura 29- Tramo rI1

Del modelo de rI1 se debe considerar un comportamiento especial para las celdas de las columnas 12 y 13 (celdas del conjunto C_{pn}), que ahora tendrán un port externo adicional que indicará la presencia o ausencia del tren (ver figura 29).

Se agregan nuevos ports de acoplamiento externo:

$\{<X_{\eta+3}(0,12), \text{binario}>, <X_{\eta+3}(1,12), \text{binario}> \} \cup \{<X_{\eta+3}(0,13), \text{binario}>, <X_{\eta+3}(1,13), \text{binario}> \}$

denotados como x-vt-tren.

Para las celdas { (0, 12), (1,12), (0,13), (1,13) } se modifican las reglas que definen el comportamiento. Estos cambios pueden consultarse en la sección 1.2.2. de [DW99].

La demora **d** = $d_{vías}$ ($d_{vías} \in N$), es una demora constante mayor que la demora que representa la velocidad normal de los autos. Esta regla representa que para poder abandonar la celda origen es necesario chequear que el tren no esté pasando (portvalue(x-vt-tren) = 0). Otro cambio que se introduce es la utilización de demora inercial fija ($d_{vías}$) más grande que la demora de las demás celdas para representar que un auto circula más lento cuando atraviesa las vías. Además esta demora permite modelar que los vehículos al llegar al paso a nivel, avanzan sólo si durante un cierto tiempo (demora inercial) tienen el camino libre (vías sin tren), caso contrario esperan a que pase el tren (desalojo de estado).

Por otro lado, las celdas que se encuentran a continuación de las vías también se ven afectadas por su presencia, pues para saber si un vehículo avanzará sobre ellas deben verificar que no haya un tren en las vías. Ellas son las celdas { (0, 13), (1,13) }.

Para poder avanzar hacia la celda origen, el vehículo que se encuentra antes de las vías, debe verificar que no esté pasando el tren (portvalue(x-vt-tren) = 0). Otro cambio que se introduce es la demora inercial fija, pues para atravesar las vías éstas deben permanecer vacías durante el tiempo que les lleva a los autos cruzarlas (La definición formal de las reglas puede consultarse en [DW99]).

Para terminar, es necesario definir cómo se hace el acoplamiento entre Trenes y Tramos.

Sea M_E el modelo de la Estación y M_{VT} el de las VíasTren. Para la tupla (rI1, 90, 1) se genera las siguientes influencias:

$I_E = \{ M_{VT} \}$,
$I_{VT} = \{ M_{rI1} \}$

El acoplamiento de los modelos se define como:

$Z_{E,VT} : \text{y-vt-tren}_E \rightarrow X_{\eta+1}(0,0)_{VT}$
$Z_{VT,rI1} : Y_{\eta+j}(0,1)_{VT} \rightarrow X_{\eta+3}(j\text{-}1,12)_{rI1} \ \forall j \ (j \in N \wedge 1 \leq j \leq 2)$
$Z_{VT,rI1} : Y_{\eta+2+j}(0,1)_{VT} \rightarrow X_{\eta+3}(j\text{-}1,13)_{rI1} \ \forall j \ (j \in N \wedge 1 \leq j \leq 2)$

En la figura 30 mostramos la implementación en la herramienta N-CD++ del Cell-DEVS que representa la vía del tren y el acoplamiento de la misma con tramos rI1, rI2 y con el Devs que representa la estación. En la figura 31 mostramos la implementación del tramo rI1 en la herramienta N-CD++.

```
[top]
components : rA rB rC rE rF rD1 rD2 rG1 rG2 rH1 rH2 rI1 rI2 via c1 c2 c3 c4
estacion@Generator
components : genAuto1@generator genAuto2@generator genAuto3@generator
genAuto4@generator
components : cont1@contador
Out : outTrans1 outTrans2
.....

% Acoplamiento via - tramo rI1
link : outPasaTren2@via inPasaTrenAntes0@rI1
link : outPasaTren2@via inPasaTrenDespues0@rI1
link : outPasaTren3@via inPasaTrenAntes1@rI1
link : outPasaTren3@via inPasaTrenDespues1@rI1
link : outPasaTren0@rI1 inPasaTren2@via
link : outPasaTren1@rI1 inPasaTren3@via

% Acoplamiento via - tramo rI2
link : outPasaTren1@via inPasaTrenAntes0@rI2
link : outPasaTren1@via inPasaTrenDespues0@rI2
link : outPasaTren0@via inPasaTrenAntes1@rI2
link : outPasaTren0@via inPasaTrenDespues1@rI2
link : outPasaTren0@rI2 inPasaTren1@via
link : outPasaTren1@rI2 inPasaTren0@via

% Acoplamiento via - estacion
link : out@Estacion inTren@via
....
....
[via]
type : cell
Width : 4
Height : 1
delay : transport
defaultDelayTime  : 10
border : nowrapped
```

Figura 30 - Implementación en la herramienta N-CD++ de la vía del tren y el acoplamiento de la misma con tramos rI1, rI2 y con el Devs que representa la estación.

Para implementar la estación se utilizó un DEVS denominado *Generator*. Este modelo genera valores cada cierto tiempo, por eso uno de sus parámetros es el tipo de distribución probabilística con que realiza esta tarea. En este caso, el modelo *Generator* representaría una estación que genera trenes (el valor 1) cada x segundos.
En la figura 31 mostramos la implementación del tramo rI1, el cuál es uno de los tramos del sector de ciudad, que es atravezado por las vías del tren.

```
[rI1]
type : cell
Width : 17
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rI1(1,-1)  rI1(1,0)  rI1(1,1)
neighbors : rI1(0,-1)  rI1(0,0)  rI1(0,1)
neighbors : rI1(-1,-1) rI1(-1,0) rI1(-1,1)
initialvalue : 0
localtransition : tramo-rule-2carril
in : inAuto0 inAuto1 inPuedeEntrar0 inPuedeEntrar1 inPasaTrenAntes0
inPasaTrenAntes1
in : inPasaTrenDespues0 inPasaTrenDespues1
link : inAuto0 inAuto@rI1(0,0)
link : inAuto1 inAuto@rI1(1,0)
link : inPuedeEntrar0 inPuedeEntrar@rI1(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rI1(1,0)
link : inPasaTrenAntes0 inPasaTren@rI1(0,12)
link : inPasaTrenAntes1 inPasaTren@rI1(1,12)
link : inPasaTrenDespues0 inPasaTren@rI1(0,13)
link : inPasaTrenDespues1 inPasaTren@rI1(1,13)
out : outPuedeEntrar0 outPuedeEntrar1  outAuto0 outAuto1
out : outPasaTren0 outPasaTren1 outQuiereCruzar0 outQuiereCruzar1
link : outPuedeEntrar@rI1(0,0) outPuedeEntrar0
```

Figura 31 - Implementación del tramo rI1 en la herramienta N-CD++

Baches

También es posible incorporar baches a algunos tramos o cruces del sector de ciudad elegido.
Esto se especifica agregando elementos al conjunto *Bache*. En este caso agregamos un bache en el tramo rA, quedando *Baches* = { (rA, 0, 10)}. Esto hace que las reglas del segmento rA permanezcan sin cambios, pero aumente la demora en la celda que tienen el bache. Esta es la celda (0,2), pues (rA, 0, 10) indica que se encuentra en el carril 0 y en la columna $\lceil 10/7{,}5 \rceil = 2$.

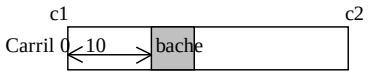


Figura 32 – Tramo con bache

También es posible incluir un bache en un cruce, por ejemplo en el cruce c1 incluimos dos baches: (c1, 0, 28) y (c1, 0, 32). Realizando los cálculos correspondientes: $\lceil 28/7{,}5 \rceil = 4$ y $\lceil 32/7{,}5 \rceil = 5$, vemos que los mismos se encuentran en las celdas (0,4) y (0,5).
.

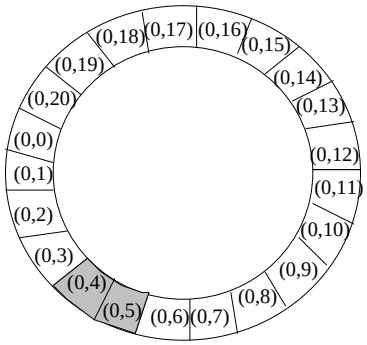


Figura 33 – Cruce con bache

Para representar un bache en el modelo Cell-Devs correspondiente al tramo (o cruce), sólo se modifica el comportamiento de la celda que lo contiene. Para ello se utiliza una demora de transporte fija suficientemente grande que refleje que los vehículos circulan despacio debido a que la calle está rota. Un bache tendrá el tamaño de una celda y para modelar la presencia de uno más grande se deben definir varios consecutivos.

Señales de Control

También es posible incorporar señales de control a algunos tramos o cruces del sector de ciudad elegido.
Esto puede especificarse agregando elementos al conjunto *Elementos de Control.* En este caso agregamos una escuela al tramo rD1, quedando *Elementos de Control*= { (rD1, escuela, 20) }. Esto hace que las reglas del segmento rD1 permanezcan sin cambios, pero aumente la demora en las celdas que tienen esta señal. Esta es la celda (0,3) pues (rD1, escuela, 20) indica que se encuentra en todos los carriles y en la columna $\lceil 20/7{,}5 \rceil = 3$.

.

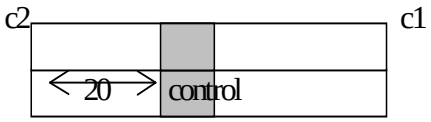


Figura 34 – Tramo rD1 con elemento de control (escuela)

La presencia de estos elementos de control afecta a los vehículos de todos los carriles de la columna donde se encuentra. Para reflejar la presencia de cualquiera de estos elementos de control en el modelo del tramo rD1, se modifica el comportamiento de las celdas de todos los carriles donde se encuentra, utilizando una demora grande y fija. Cabe destacar que el objetivo de estos elementos de control es que los vehículos disminuyan la velocidad, y esto se logra aumentando la demora de las celdas. El tipo de demora (transporte o inercial) no cambia pero según el elemento de control se definirán distintas duraciones.

Nota: En esta sección mostramos parte de la implementación del sector de ciudad de la figura 1, en la herramienta N-CD++. La implementación completa de este sector puede consultarse en el apéndice B.

Ejecución de simulaciones sobre el sector de ciudad elegido

Como comentamos anteriormente, el sector de ciudad mostrado en la figura 1 fue implementado en la herramienta N-CD++ [RW99]. Esto nos permitió efectuar varias pruebas variando ciertos parámetros como por ejemplo la frecuencia de partida de trenes de la estación, o la cantidad de autos que ingresan por segundo a la simulación, representando el flujo de tráfico en una hora pico y en una hora no pico. En la figura 35, mostramos el log de mensajes de una de las corridas realizadas y la figura 36 muestra parte del resultado de una de las corridas (tramo rI1 y cruce c1), que fue generada usando la opción de graficación de la herramienta.

```
Mensaje I / 00:00:00:000 / Root(00) para top(01)
Mensaje I / 00:00:00:000 / top(01) para ra(02)
Mensaje I / 00:00:00:000 / top(01) para rb(21)
Mensaje I / 00:00:00:000 / top(01) para rc(32)
Mensaje I / 00:00:00:000 / top(01) para re(47)
Mensaje I / 00:00:00:000 / top(01) para rf(62)
Mensaje I / 00:00:00:000 / top(01) para rd1(77)
Mensaje I / 00:00:00:000 / top(01) para rd2(112)
Mensaje I / 00:00:00:000 / top(01) para rg1(147)
Mensaje I / 00:00:00:000 / top(01) para rg2(256)
Mensaje I / 00:00:00:000 / top(01) para rh1(365)
Mensaje I / 00:00:00:000 / top(01) para rh2(394)
Mensaje I / 00:00:00:000 / top(01) para ri1(423)
Mensaje I / 00:00:00:000 / top(01) para ri2(460)
Mensaje I / 00:00:00:000 / top(01) para via(497)
Mensaje I / 00:00:00:000 / top(01) para c1(502)
Mensaje I / 00:00:00:000 / top(01) para c2(524)
Mensaje I / 00:00:00:000 / top(01) para c3(531)
Mensaje I / 00:00:00:000 / top(01) para c4(535)
Mensaje I / 00:00:00:000 / top(01) para estacion(538)
Mensaje I / 00:00:00:000 / top(01) para genauto1(539)
Mensaje I / 00:00:00:000 / top(01) para genauto2(540)
Mensaje I / 00:00:00:000 / top(01) para genauto3(541)
Mensaje I / 00:00:00:000 / top(01) para genauto4(542)
Mensaje I / 00:00:00:000 / top(01) para cont1(543)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,0)(03)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,1)(04)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,2)(05)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,3)(06)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,4)(07)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,5)(08)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,6)(09)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,7)(10)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,8)(11)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,9)(12)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,10)(13)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,11)(14)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,12)(15)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,13)(16)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,14)(17)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,15)(18)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,16)(19)
Mensaje I / 00:00:00:000 / ra(02) para ra(0,17)(20)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,0)(22)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,1)(23)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,2)(24)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,3)(25)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,4)(26)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,5)(27)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,6)(28)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,7)(29)
Mensaje I / 00:00:00:000 / rb(21) para rb(0,8)(30)
```

Figura 35 – Log de mensajes

Cruce c1

```
....

Line : 9763 - Time: 00:00:00:480
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
    +-------------------------------------------------------------------------------------+
   0|                  1   1   1   1                                                      |
    +-------------------------------------------------------------------------------------+
Line : 10003 - Time: 00:00:00:490
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
    +-------------------------------------------------------------------------------------+
   0|                  1   1   1       1                                              1   1|
    +-------------------------------------------------------------------------------------+
Line : 10276 - Time: 00:00:00:500
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
    +-------------------------------------------------------------------------------------+
   0|                  1   1       1                                                  1    |
    +-------------------------------------------------------------------------------------+
Line : 10556 - Time: 00:00:00:510
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
    +-------------------------------------------------------------------------------------+
   0|                  1       1                                                          1|
    +-------------------------------------------------------------------------------------+
Line : 10792 - Time: 00:00:00:520
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
    +-------------------------------------------------------------------------------------+
   0|                      1       1                                                      |
    +-------------------------------------------------------------------------------------+
....
```

Tramo rI1

```
....
Line : 1352 - Time: 00:00:00:010
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
    +-------------------------------------------------------------------------+
   0|   1                                                                     |
   1|   1                                                                     |
    +-------------------------------------------------------------------------+
Line : 1490 - Time: 00:00:00:020
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
    +-------------------------------------------------------------------------+
   0|       1                                                                 |
   1|       1                                                                 |
    +-------------------------------------------------------------------------+
Line : 1632 - Time: 00:00:00:030
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
    +-------------------------------------------------------------------------+
   0|           1                                                             |
   1|           1                                                             |
    +-------------------------------------------------------------------------+
Line : 1798 - Time: 00:00:00:040
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
    +-------------------------------------------------------------------------+
   0|               1                                                         |
   1|               1                                                         |
    +-------------------------------------------------------------------------+
Line : 1940 - Time: 00:00:00:050
        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
    +-------------------------------------------------------------------------+
   0|                   1                                                     |
   1|                   1                                                     |
    +-------------------------------------------------------------------------+
....
```

Figura 36 – Resultado de una de las corridas realizadas (ejemplo hora pico). En la figura se observa parte del resultado del tramo rI1 y del cruce c1.

Fueron realizadas varias simulaciones, intentando representar las siguientes situaciones:

Hora No pico: representa el flujo de tráfico en el sector de ciudad elegido en una hora no pico.

Hora pico: representa el flujo de tráfico en el sector de ciudad elegido en una hora pico.

Existencia de un bache en cruce c1: El cruce c1 se encuentra conectado a nueve tramos. La existencia de un bache en el mismo podría influir notoriamente en el flujo de vehículos del sector.

Aumento del tiempo en que la barrera (que atraviesa los tramos rI1 y rI2) permanece baja: Esto influiría también en el flujo de vehículos del sector, aumentado principalmente la congestión en los tramos rI1 y rI2

Al realizar la implementación del sector de ciudad elegido en la herramienta N-CD++ y al efectuar las distintas pruebas mencionadas, detectamos algunos problemas y bugs en la herramienta. El detalle de los mismos se encuentra descripto en el apéndice C.

En todas las pruebas realizadas el tiempo de simulación fue de 10 segundos. Para realizar las distintas simulaciones se inyectaron autos, en distintos puntos del modelo utilizando generadores con distribución constante a distintas frecuencias. Se utilizaron cuatro generadores de autos: un generador que inyecta autos en el tramo rG1 (generador 1), un generador que inyecta autos en el tramo rH2 (generador 2), un generador que inyecta autos en el tramo rA (generador 3) y por último un generador que inyecta autos en el tramo rI2 (generador 4). En la figura 37 puede observarse el sector de ciudad y los generadores.

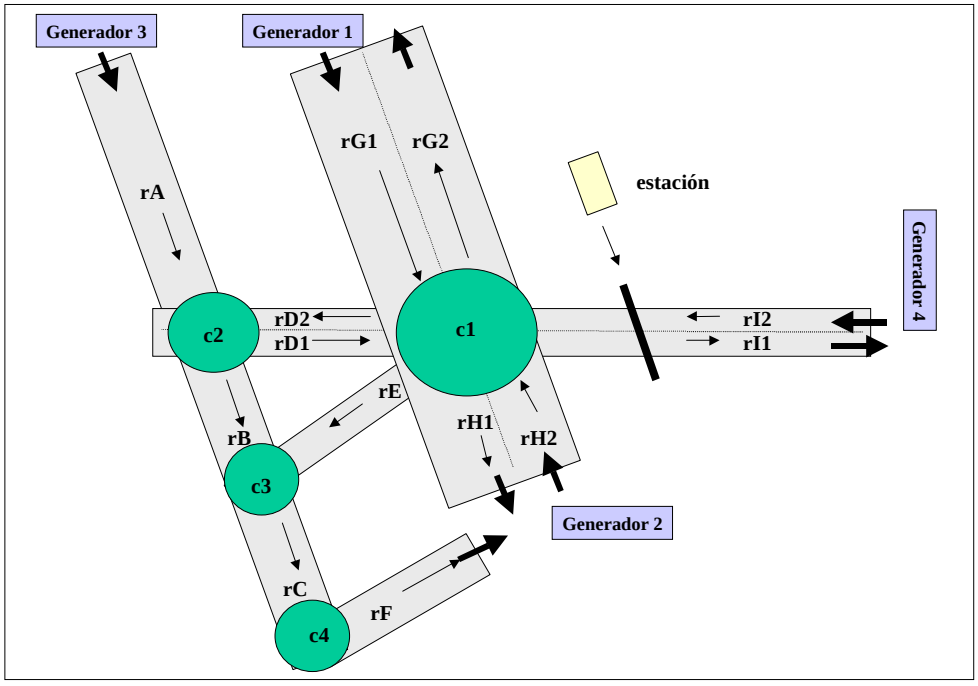


Figura 37 – Sector de ciudad y generadores

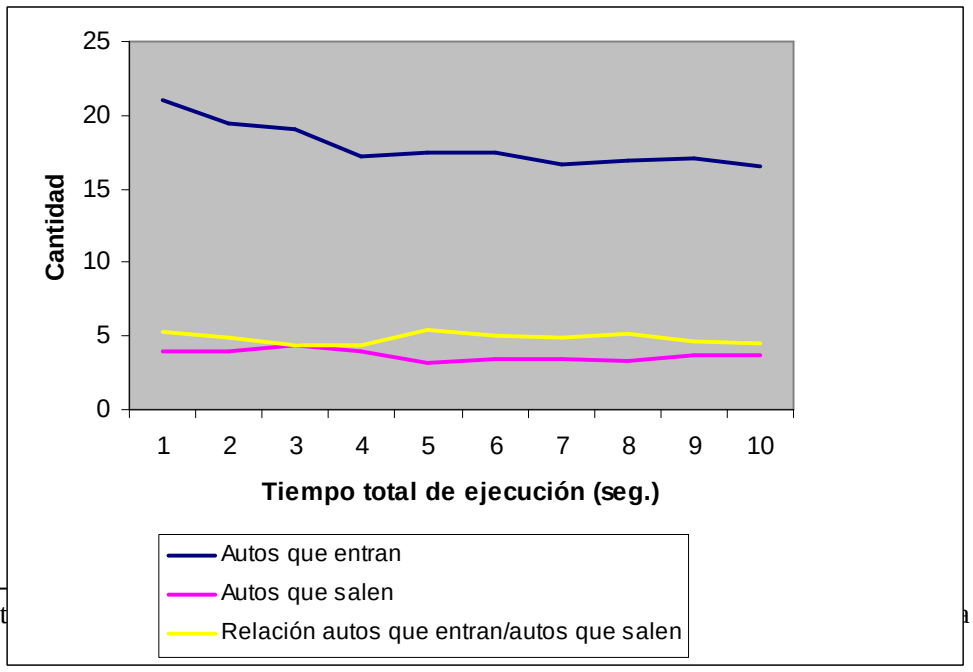
1) Simulación Hora no pico:

A continuación mostramos los parámetros utilizados para realizar esta simulación:

Demora autos: 10
Demora ferrocarril: 50
Frecuencias generadores:
Generador tramo rG1 : 0.7 seg.
Generador tramo rH2 : 0.7 seg.
Generador tramo rA : 0.5 seg.
Generador tramo rI2 : 0.6 seg.

La figura muestra los resultados obtenidos: cantidad de autos inyectados en la simulación por unidad de tiempo (seg), cantidad de autos que salen de la simulación por unidad de tiempo (seg) y por último la relación entre estos dos valores. Este último resultado nos estaría indicando cuán congestionado estuvo el sector de ciudad durante el tiempo que duró la simulación: así si entran muchos autos y salen pocos en un determinado intervalo de tiempo podríamos concluir que el sector presenta congestión, demorando la salida de los vehículos.

Tiempo	Autos que entran	Autos que salen	Relación autos que entran/autos que salen
00:00:01:000	21,0000	4,0000	5,2500
00:00:02:000	19,5000	4,0000	4,8750
00:00:03:000	19,0000	4,3333	4,3846
00:00:04:000	17,2500	4,0000	4,3125
00:00:05:000	17,4000	3,2000	5,4375
00:00:06:000	17,50000	3,50000	5,0000
00:00:07:000	16,7143	3,42857	4,8750
00:00:08:000	16,875	3,25	5,1923
00:00:09:000	17	3,66667	4,6364
00:00:10:000	16,5	3,7	4,4595



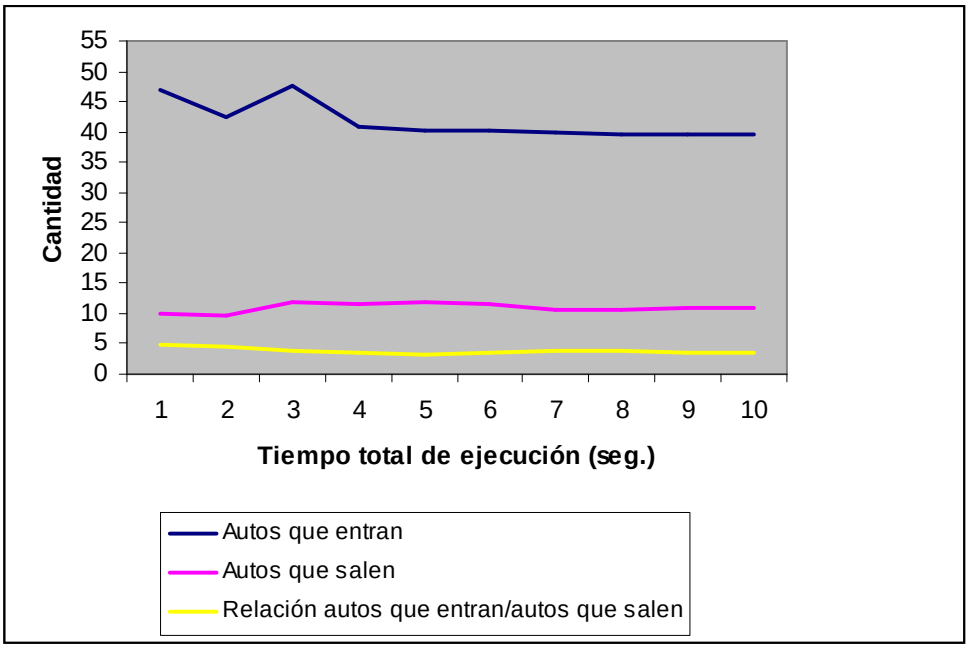
2) Simulación Hora pico:

A continuación mostramos los parámetros utilizados para realizar esta simulación. En este caso, para simular una hora pico se inyectaron mayor cantidad de autos por unidad de tiempo, es decir variamos la frecuencia de los generadores, principalmente los correspondientes a los tramos rG1 y rH2, ya que en la realidad, es en estos tramos en donde se incrementa mayormente el flujo de vehículos en las horas pico.

Demora autos: 10
Demora ferrocarril: 50
Frecuencias generadores:
Generador tramo rG1 : 0.2 seg.
Generador tramo rH2 : 0.2 seg.
Generador tramo rA : 0.5 seg.
Generador tramo rI2 : 0.3 seg.

Las siguientes figuran muestran los resultados correspondientes a dicha simulación.

Tiempo	Autos que entran	Autos que salen	Relación Autos que entran/autos que salen
00:00:01:000	47,0000	10,0000	4,7000
00:00:02:000	42,5000	9,5000	4,4737
00:00:03:000	47,6667	12,0000	3,9722
00:00:04:000	40,7500	11,5000	3,5435
00:00:05:000	40,2000	12,0000	3,3500
00:00:06:000	40,16670	11,50000	3,4928
00:00:07:000	39,8571	10,5714	3,7703
00:00:08:000	39,625	10,625	3,7294
00:00:09:000	39,6667	10,7778	3,6804
00:00:10:000	39,5	10,9	3,6239



En el gráfico se observa que la cantidad de autos que entran por segundo en este ejemplo (hora pico) es mayor a la cantidad de autos que entran por segundo en el ejemplo anterior (hora no pico). Esto mismo sucede con la cantidad de autos que salen del modelo por segundo.

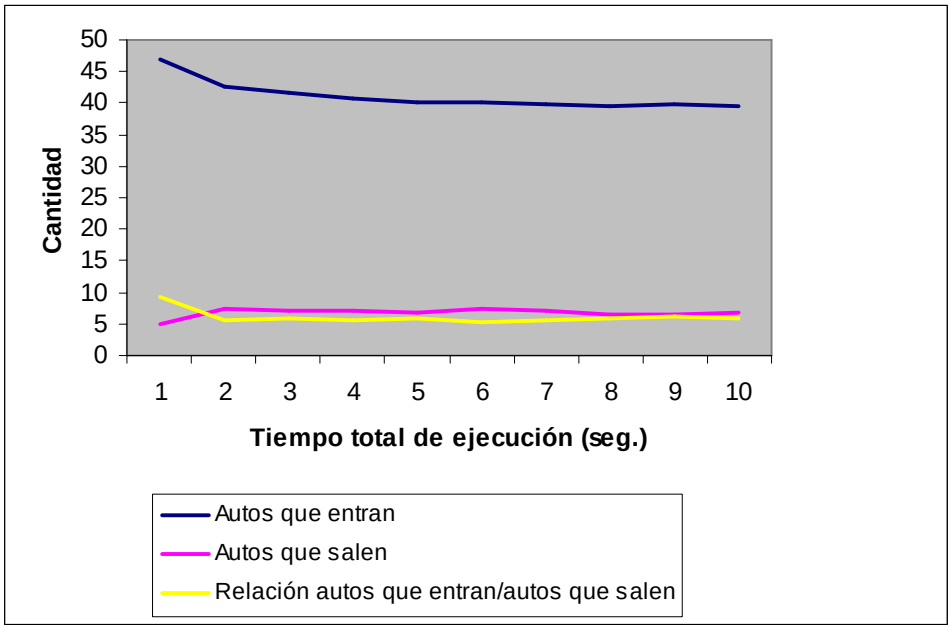
3) Existencia de un bache en cruce c1:

A continuación mostramos los parámetros utilizados para realizar esta simulación. En este caso, se incluyó en la definición del sector de ciudad, un bache en el cruce c1. El bache se introdujo en las celdas 4 y 5 del cruce c1. En las celdas en donde se encuentra el bache la demora es mayor que en el resto de las celdas, simulando que los autos circulan más despacio debido a la existencia de un bache. Las frecuencias de los generadores se mantuvieron igual al del ejemplo de hora pico.

Demora autos: 10
Demora ferrocarril: 50
Demora en celdas del bache : 800
Frecuencias generadores:
Generador tramo rG1 : 0.2 seg.
Generador tramo rH2 : 0.2 seg.
Generador tramo rA : 0.5 seg.
Generador tramo rI2 : 0.3 seg.

Las siguientes figuran muestran los resultados correspondientes a dicha simulación.

Tiempo	Autos que entran	Autos que salen	Relación autos que entran/autos que salen
00:00:01:000	47,0000	5,0000	9,4000
00:00:02:000	42,5000	7,5000	5,6667
00:00:03:000	41,6667	7,0000	5,9524
00:00:04:000	40,7500	7,2500	5,6207
00:00:05:000	40,2000	6,8000	5,9118
00:00:06:000	40,16670	7,50000	5,3556
00:00:07:000	39,8571	7	5,6939
00:00:08:000	39,625	6,625	5,9811
00:00:09:000	39,6667	6,55556	6,0508
00:00:10:000	39,5	6,7	5,8955



En el gráfico se observa que la cantidad de autos que ingresan al modelo por segundo en este ejemplo, es similar a la del ejemplo anterior. Sin embargo la cantidad de autos que salen del modelo por segundo es menor. Esto se debe a la presencia de un bache en el cruce c1, que ocasiona congestión demorando la salida de vehículos del modelo.

4) Aumento del tiempo en que la barrera (que atraviesa los tramos rI1 y rI2) permanece baja:

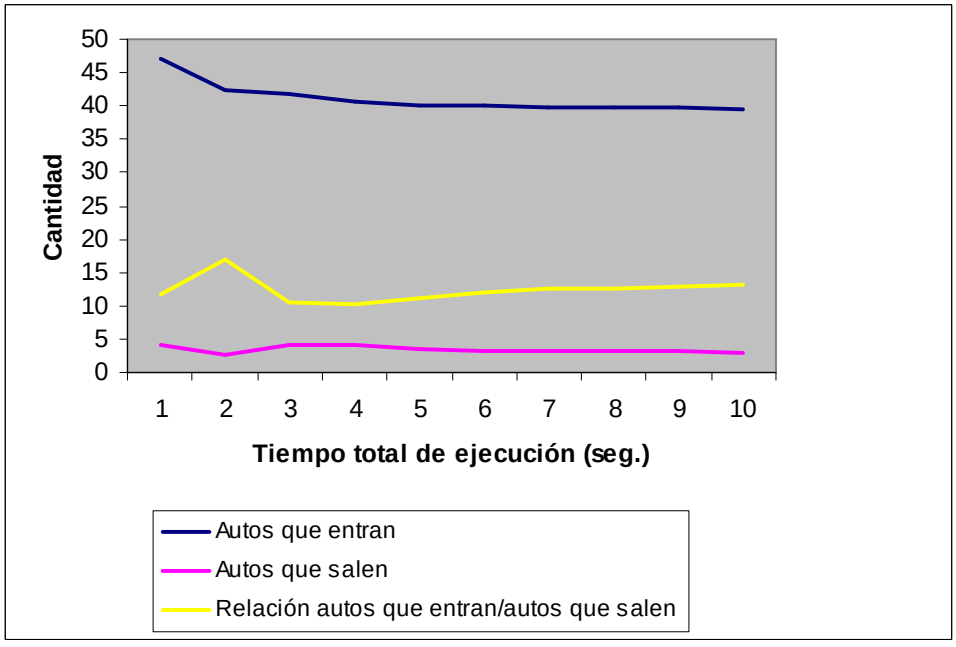
A continuación mostramos los parámetros utilizados para realizar esta simulación. En este caso, para simular el tiempo en que la barrera permanece baja se aumentó la demora en las celdas del Cell-Devs que representa las vías del tren (antes 50, ahora 500).

Cabe señalar que en este ejemplo no se incluyo el bache y que las frecuencias de los generadores se mantuvieron igual al del ejemplo de hora pico.

Demora autos: 10
Demora ferrocarril: 500
Frecuencias generadores:
Generador tramo rG1 : 0.2 seg.
Generador tramo rH2 : 0.2 seg.
Generador tramo rA : 0.5 seg.
Generador tramo rI2 : 0.3 seg.

Las siguientes figuran muestran los resultados correspondientes a dicha simulación:

Tiempo	Autos que entran	Autos que salen	Relación autos que entran/autos que salen
00:00:01:000	47,0000	4,0000	11,7500
00:00:02:000	42,5000	2,5000	17,0000
00:00:03:000	41,6667	4,0000	10,4167
00:00:04:000	40,7500	4,0000	10,1875
00:00:05:000	40,2000	3,6000	11,1667
00:00:06:000	40,16670	3,33333	12,0500
00:00:07:000	39,8571	3,14286	12,6818
00:00:08:000	39,625	3,125	12,6800
00:00:09:000	39,6667	3,11111	12,7500
00:00:10:000	39,5	3	13,1667



En el gráfico se observa que la cantidad de autos que ingresan al modelo por segundo en este ejemplo, es similar a la del ejemplo 2 (hora pico). Sin embargo la cantidad de autos que salen del modelo por segundo es menor. Esto se debe a que en este ejemplo el tiempo en que la barrera

permanece baja es mayor al del ejemplo 2, lo que provoca una mayor congestión y una mayor demora en la salida de vehículos del modelo.

Cabe señalar que podrían obtenerse mejores resultados utilizando generadores con distribuciones aleatorias. En un principio intentamos implementar el ejemplo del sector de ciudad utilizando este tipo de generadores, pero se nos presentaron algunos inconvenientes que serán descriptos en el apéndice C.

Sección IV: Extensiones al lenguaje de especificación

Al lenguaje de especificación presentado en [DW99] se le han agregado las siguientes extensiones:

Ruteo: Los modelos definidos en [DW99] no modelan que los vehículos parten de un determinado origen hacia un destino particular, sino que el camino de los mismos se va construyendo en forma aleatoria. Esta definición no modela un aspecto esencial de la realidad, como es la trayectoria de los autos.
En los modelos propuestos en este trabajo, se incluyó el camino como una característica del vehículo. Para ello, fue necesario definir una estructura auxiliar a los modelos Cell-Devs, pero que forma parte de la simulación, que contenga información de ruteo entre cada punto de la sección de ciudad modelada. Esta estructura es una matriz Origen-Destino (OD).

Monitoreo de congestión: Se ha tenido en cuenta, la importancia de poseer información acerca de cuán congestionada se encuentra la sección modelada y cuales son sus puntos conflictivos. Esta información es útil, por ejemplo, para evaluar alternativas estructurales que permitan solucionar los problemas que causan los puntos de congestión, o brindarle información a los conductores que les permita tomar decisiones.
En este trabajo, se han incorporado modelos que almacenan información de congestión, que luego es utilizada para tomar decisiones de ruteo. Sin embargo, como ya se ha dicho, esta información también puede ser utilizada con otros propósitos.

Comportamiento del conductor: Este aspecto, es muy importante a la hora de modelar las decisiones que toman los conductores. Este es un tema muy vasto ya que determina todos los movimientos de los vehículos (velocidad a la cual se mueve el auto, decisión de cambio de carril, etc). En este trabajo, se presentan varios aspectos a tener en cuenta, pero sólo se detalla el que caracteriza la velocidad a la cual se mueven los vehículos, modificando la definición de los modelos Cell-Devs.

A continuación se explicará en forma más detallada las extensiones y cambios propuestos.
Luego, en cada sección se especifica el mapeo del lenguaje de especificación con los modelos DEVS y Cell-Devs.

Matriz Origen Destino (OD)

Las matrices Origen destino (OD) son importantes para describir el transporte de una región. Estas matrices contienen información sobre rutas y transporte entre diferentes zonas o puntos de una región.
Existen numerosos métodos para estimar matrices OD. A partir de la definición de una región, representada por un grafo, los métodos se basan en información disponible, tal como la cantidad de tráfico o la demora existente en cada enlace del grafo.
Dependiendo del objetivo para el cual se usará esta matriz será conveniente un determinado método de estimación.
La obtención de este tipo de matrices y los algoritmos utilizados para tomar decisiones de ruteo constituyen, por sí solos, todo un trabajo de investigación. Esta tesis no se centra en este problema, sino que asume que se conoce la matriz OD (la cual ha sido obtenida o estimada por algún método existente) y que es utilizada por la simulación para tomar decisiones acerca del ruteo de los vehículos.

De la bibliografía consultada se ha tomado como ejemplo dos estructuras de datos que pueden ser utilizadas para representar a la matriz OD. Sin embargo, esta definición es accesoria a los modelos de simulación presentados en este trabajo, pudiendo intercambiarse por cualquier otra estructura o algoritmos que se consideren convenientes de acuerdo al caso que se esté simulando.

A continuación serán descriptas, los dos tipos de matrices OD:

1) ***Tabla de Caminos:*** Cada registro de la tabla de caminos especifica un camino que conecta a un determinado par origen-destino y el tiempo que se tarda llendo por ese camino. Pueden existir distintas tablas de caminos según distintos parámetros como el momento del día y el tipo de vehículo, ya que el contenido de la matriz puede variar sustancialmente de acuerdo a estos parámetros.

   Tiempo Tipo de Vehículo
   { ID Nodo origen Nodo destino {Enlace-1 Enlace-2…. Enlace-N} Tiempo de viaje }

   A continuación mostramos un ejemplo de una matriz OD que presenta este formato, cuya información se basa en el grafo de la figura 1, el cual representa una red de transporte.

06:00 AM - 09:00 AM - Autos particulares
{
{ 1 14 15 {12 77 7 47 18} 94 }
{ 2 14 15 {12 23 73 3 43 21 18} 117.3 }
{ 3 14 20 {12 23 73 3 43 15} 109.6 }
{ 4 14 20 {12 77 7 47 25 15} 110.9 }
{ 5 14 30 {12 23 14} 28.9 }
{ 6 15 14 {17 71 1 41 11 } 93.3 }
{ 7 15 14 {17 25 75 5 45 27 11} 115.5 }
{ 8 15 20 {17 25 15} 26.3 }
{ 9 15 30 {17 25 75 5 45 14} 110.8 }
{ 10 15 30 {17 71 1 41 23 14} 111.3 }
{ 11 20 14 {16 21 71 1 41 11} 109.7 }
{ 12 20 14 {16 75 5 45 27 11} 109.8 }
{ 14 20 30 {16 21 71 1 41 23 14} 128.3 }
{ 15 20 30 {16 75 5 45 14} 105.4 }
{ 16 30 14 {13 27 11} 28.9 }
{ 17 30 15 {13 27 77 7 47 18} 112.4 }
{ 18 30 15 {13 73 3 43 21 18} 113.1 }
{ 19 30 20 {13 73 3 43 15} 105 }
{ 20 30 20 {13 27 77 7 47 25 15} 126.9}
}

   En las simulaciones que utilizan este tipo de matrices, la información de ruteo es estática.
   Existen determinados caminos, con sus respectivos tiempos de recorrido, para cada par origen – destino, los cuáles se mantienen durante todo el transcurso de la simulación.
   Por ello, el cálculo de esta matriz puede realizarse antes del comienzo de la simulación y sólo se usa durante la ejecución de la misma.

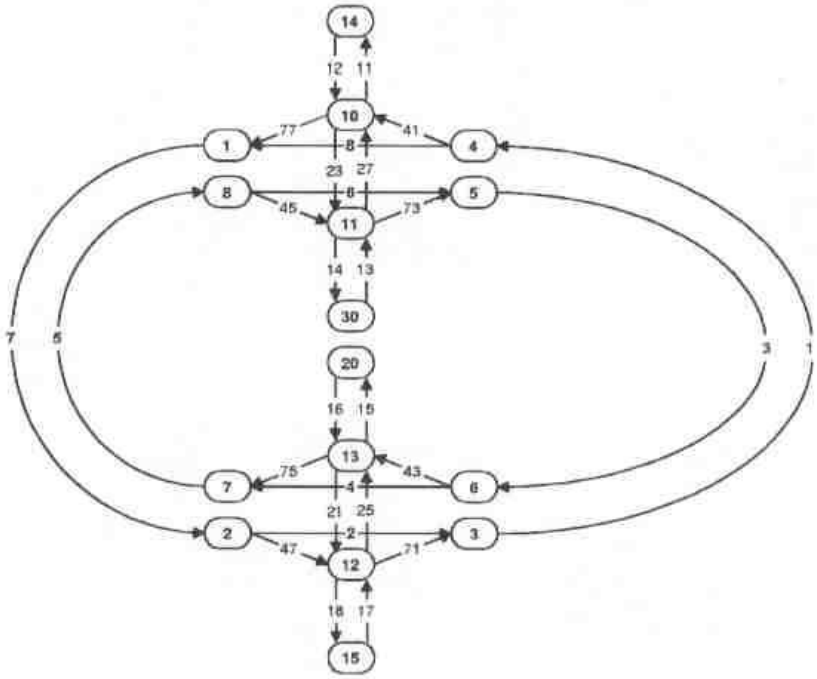


Figura 1 - Red de transporte

2) ***Tabla de Enlaces:*** Esta tabla contiene información sobre el flujo de tráfico o la demora de los distintos enlaces que conforman la red de transporte. Pueden existir distintas tablas según distintos parámetros como el momento del día y el tipo de vehículo.

Tiempo Tipo de Vehículo
{ Enlace flujo/demora
}

A continuación mostramos un ejemplo de una matriz OD que presenta este formato:

PURPOSE: All
MODE: All
PERIOD: 24-Hour

	TO:						
FROM:	**METRO TORONTO**	**DURHAM**	**YORK**	**PEEL**	**HALTON**	**HAMILTON-WENTWORTH**	**REGION TOTAL**
METRO TORONTO	3,726,400	64,700	219,700	234,600	33,800	10,300	**4,289,500**
DURHAM	65,300	520,200	13,000	3,600	600	500	**603,200**
YORK	219,700	12,900	400,300	22,100	2,400	800	**658,200**
PEEL	234,300	3,700	22,900	774,700	44,500	7,200	**1,087,200**
HALTON	34,300	500	2,400	43,900	429,700	49,100	**559,900**
HAMILTON-WENTWORTH	10,500	400	800	7,100	49,100	771,200	**839,100**
REGION TOTAL	**4,290,500**	**602,400**	**659,000**	**1,086,100**	**560,100**	**839,100**	**8,037,600**

Figura 2 - Ejemplo de matriz OD

Este tipo de matriz OD permite que los vehículos varíen su recorrido durante el transcurso de la simulación de acuerdo al estado de los enlaces. La información contenida en la matriz OD se irá actualizando durante la simulación, permitiendo de esta manera disponer de información más precisa para la toma de decisiones.

Estimación de matrices OD

Para estimar cualquiera de los dos tipos de matrices OD descriptas anteriormente, se utilizan algoritmos sobre grafos, que determinan caminos que minimicen el tiempo de viaje entre cualquier par de puntos de la red de transporte.

Existe un gran número de algoritmos para resolver este tipo de problemas conocidos en su conjunto como *"Problemas de caminos más cortos"*.

Para la primer alternativa (tabla de caminos) nos interesan varios caminos para cada cada par de puntos, con su respectivo tiempo de viaje.
El problema de hallar los *k* mejores caminos entre dos nodos es NP-Hard [Gar/79], con lo cual no se conocen algoritmos polinomiales para resolverlo. Sigue siendo NP-Hard aún cuando se consideran ejes de peso 1, en grafos dirigidos y no dirigidos. Sin embargo, el problema puede resolverse en tiempo pseudopolinomial (polinomial en *n, k,* y log *B,* siendo *n* la cantidad de nodos de la red y *B* el peso máximo entre todos los ejes), con lo cual es resoluble en tiempo polinomial para cualquier valor *fijo* de *k* [Joh/76].

Aunque el problema general de hallar los mejores *k* caminos entre dos nodos es NP-Hard, es posible hallar un algoritmo polinomial para el caso $k = 2$, es decir, buscar el segundo camino mínimo entre *s* y *t* en una red *G*. El siguiente pseudocódigo describe el algoritmo:

Buscar el camino mínimo *P* entre *s* y *t* en *G*.

Para cada eje $ij \in P$:
Buscar el camino P_{ij} mínimo entre *s* y *t* en $G \setminus \{ij\}$.

El segundo camino más corto entre *s* y *t* es el camino P_{ij} de costo menor entre todos los caminos P_{rs}, con $rs \in P$ (es decir, entre todos los caminos generados en el ciclo anterior).

Para verificar la correctitud del algoritmo, basta observar que el segundo mejor camino *P'* entre *s* y *t* debe diferir de *P* en al menos un eje. Sea C_{ij} el conjunto de caminos entre *s* y *t* que no incluyen al eje *ij*. Entonces, $|P'| = \min \{ |Q| : Q \in \cup_{ij \in P} C_{ij} \}$, donde $|Q|$ denota el costo total del camino *Q*. Ahora, como $|P_{ij}| = \min \{ |Q| : Q \in C_{ij} \}$, entonces

$$|P'| = \min \{ |Q| : Q \in \cup_{ij \in P} C_{ij} \} = \min \{ |P_{ij}| : ij \in P \}.$$

Por lo tanto, el camino *P'* hallado es el segundo camino mínimo entre *s* y *t*. El tiempo de ejecución de este algoritmo es de $O(n\ S(G))$, donde $S(G)$ es el tiempo de ejecución de cualquier algoritmo para calcular caminos mínimos en una red.

Supóngase un grafo dirigido G= {V, A} en el cual cada arco tiene una etiqueta no negativa, y donde un vértice se especifica como origen. El problema es determinar el costo del camino más corto del origen a todos los demás vértices de V, donde la longitud de un camino es la suma de los costos de los arcos del camino. Esto se conoce con el nombre de problema de los caminos más cortos con un solo origen. Obsérvese que se hablará de caminos con <<longitud>> aun cuando los costos representan algo diferente, como tiempo.
El algoritmo opera a partir de un conjunto S de vértices cuya distancia más corta desde el origen ya es conocida. En principio, S contiene solo el vértice de origen. En cada paso, se agrega algún vértice restante v a S, cuya distancia desde el origen es la mas corta posible. Suponiendo que todos los arcos tienen costo no negativo, siempre es posible encontrar un camino más corto entre el origen y v que pasa solo a través de los vértices de S, y que se llama especial. En cada paso del algoritmo, se utiliza un arreglo D para registrar la longitud del camino especial más corto a cada vértice. Una vez que S incluye todos los vértices, todos los caminos son <<especiales>>, así que D contendrá la distancia mas corta del origen a cada vértice.
El algoritmo se muestra en la siguiente figura, donde se supone que existe un grafo dirigido G= {V, A} en el que V= {1,2,….,n} y el vértice 1 es el origen. C es un arreglo bidimensional de costos, donde C[i, j] es el costo de ir del vértice i al vértice j por el arco i→j. Si no existe el arco i→j, se supone que C[i,j] es ∞, un valor mucho mayor que cualquier costo real. En cada paso D[i] contiene la longitud del camino especial mas corto actual para el vértice i.

```
Procedure Dijkstra;
Begin
        S:={1};
        For i:= 2 to n do
                D[i]:= C[1, i]; {asigna valor inicial a D}

        For i:=1 to n-1 do begin
                Elige un vértice w en V-S tal que D[w] sea un mínimo;
                Agrega w a S;
                For cada vértice v en V-S do
                        D[v]:= min(D[v], D[w] + C[w, v])
        End
End; {Dijsktra}
```

Para la segunda alternativa, podría utilizarse el *algoritmo de Dijkstra* (ver figura 3) que calcula los caminos más cortos desde un nodo dado a todos los demás nodos de la red. El algoritmo es aplicable al problema dado que todos los ejes de la red tienen peso no negativo, y requiere un tiempo de ejecución de O(*m log n*), siendo *n* y *m* la cantidad de nodos y de ejes de la red, respectivamente.

Figura 3 -Algoritmo de Dijkstra

Como en este caso solamente necesitamos calcular el camino mínimo desde *s* hasta *t*, en lugar de hacerlo hacia todos los otros nodos, podemos considerar una modificación al algoritmo básico de Dijkstra, llamado *Algoritmo de Dijkstra bidireccional*, que realiza esta tarea [Hel/88]. En este algoritmo, aplicamos simultáneamente el algoritmo de Dijkstra desde el nodo *s* y el algoritmo reverso de Dijkstra desde el nodo *t*. Del mismo modo que en el esquema básico, el algoritmo construye conjuntos *S* y *S'* desde *s* y *t*, respectivamente, hasta que ambos esquemas marcan algún

nodo como permanente (es decir, $S \cap S' = \{k\}$). En este punto, se puede construir un camino mínimo entre *s* y *t*, según se muestra más adelante.

El algoritmo puede describirse mediante el siguiente pseudocódigo:

S, S' ← ∅
d(i), d'(i) ← ∞ para todos los nodos i
d(s), d'(t) ← 0
paso ← 1

Mientras S ∩ S' = ∅:

Si paso es par:
Seleccionar un nodo i ∉ S tal que d(i) = min {d(j) : j∉ S } y agregarlo a S
Para cada eje ij, si d(j) > d(i) + c_{ij} entonces d(j) ← d(i) + c_{ij}, pred(j) ← i

Si paso es impar
Seleccionar un nodo i ∉ S' tal que d'(i) = min {d'(j) : j∉ S } y agregarlo a S'
Para cada eje ji, si d'(j) > d'(i) + c_{ij} entonces d'(j) ← d'(i) + c_{ij}, suc(j) ← i

paso ← paso + 1;

Fin (mientras);

El algoritmo construye dos conjuntos *S* y *S'* partiendo desde *s* y *t* respectivamente. Además, mantiene un conjunto de índices *pred* indica el predecesor de cada nodo *i* de *S* en un camino mímimo *P(i)* desde *s* hasta ese nodo. Del mismo modo, el índice *suc* indica el sucesor desde cada nodo *i* en un camino mínimo *P'(i)* hasta *t*. Esta propiedad se mantiene invariante a lo largo de la ejecución.

Cuando el algoritmo de Dijkstra bidireccional termina, entonces S ∩ S' = {k} para algún nodo k. Es posible probar que el camino más corto entre s y t es o bien P(k) ∪ P'(k), o bien P(i) ∪ {ij} ∪ P'(j), para algún eje ij con i ∈ S, y j ∈ S' (ver [Hel/88]). Aunque el tiempo de ejecución tiene un peor caso similar al del algoritmo básico, este nuevo método es muy eficiente, dado que tiende a visitar una cantidad reducida de nodos [Ahu/93].

Sin embargo, surgen ciertas complicaciones cuando se aplican los algoritmos de caminos mas cortos, como los dos discutidos anteriormente, al problema de tráfico de una ciudad.

Por ejemplo, pueden surgir algunos problemas, si se desea tomar en cuenta los giros (en términos de incremento de tiempo de viaje) que realiza un conductor en una red de transporte. Consideremos, por ejemplo la red de la figura 4 que representa una parte pequeña de una ciudad. Los números cercanos a los arcos indican el tiempo promedio de viajes en cada segmento y los nodos representan las esquinas. Así el camino mas corto del nodo A al nodo D es el camino {A, B, D} de longitud 12 (incluyendo las dos unidades de infracción por doblar en B), mientras que el camino más corto de A a E es {A, C, D, E} de longitud 20. Así, el camino que un conductor seguiría para ir de A a D, siguiendo el camino más corto para ir de A a E, no coincide con (y es de diferente longitud) el camino más corto desde A a D!.

Este ejemplo sugiere que los modelos de red (como el del ejemplo de la figura) deben ser modificados de alguna manera, para que puedan ser aplicados los algoritmos de caminos mas cortos descriptos anteriormente, cuando se quiere tomar en cuenta los giros que realizan los automovilistas. Además, los caminos más cortos en una ciudad podrían incluir ciclos (en el sentido de pasar por la misma esquina dos veces). Por ejemplo, si los giros a la izquierda estuvieran prohibidos en la intersección D de la figura, el camino mas corto desde E a F sería E,D.B,A, C,D,F}. Esto, otra vez, no es contemplado por los algoritmos de ruteo mencionados anteriormente.

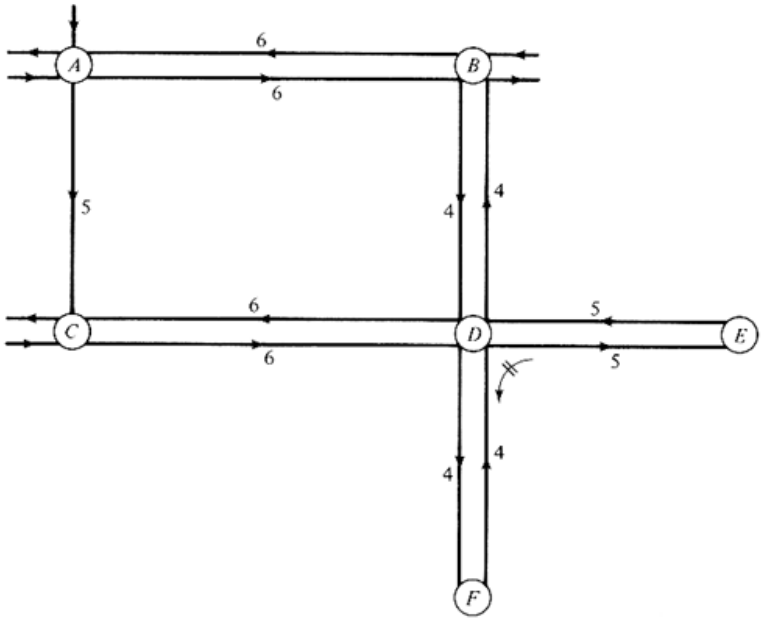


Figura 4 - Ejemplo de red de transporte

Matrices OD utilizadas

En este informe utilizamos matrices OD, del tipo Tabla de caminos, ya que es la opción cuya implementación nos resultó más sencilla.

Como ya dijimos, esta tesis se centra en la construcción de modelos de simulación que tengan en cuenta la información de ruteo, sin concentrarnos en obtener la mejor forma de encontrar el camino más adecuado.
Una vez definido el modelo de simulación, la estructura de la matriz OD y la función que toma las decisiones de ruteo pueden ser redefinidas sin afectar al modelo de simulación. Esto nos permite tratar ambos problemas en forma absolutamente independiente.

La interrelación entre la especificación en el lenguaje ATLAS, la información de ruteo y los modelos Cell-Devs es la siguiente:

1. Se especifican los modelos en el lenguaje de especificación ATLAS.
2. Como cada tramo y cada cruce poseen una identificación, y se especifica la interconexión de los mismos, se puede construir a partir de ella un grafo dirigido (en caso de poseerse, se podría agregar a la información de cada tramo información estadística, pudiendo asignarse un peso a cada enlace del grafo)
3. A partir del grafo, se puede construir la matriz OD.
4. Los modelos de simulación que utilizan la información de ruteo hacen uso de una función que consulta a la matriz OD, para dar su respuesta.

Ruteo

En esta sección, se agregará a los modelos con cruces existentes definidos en [DW99] información acerca de origen - destino de cada vehículo. Se utilizarán matrices origen – destino (OD) y algoritmos de ruteo para determinar los posibles caminos a seguir para cada par origen – destino.

Esto permitirá entre otras cosas, si se tiene información estadística sobre puntos de concentración urbana de una ciudad, modelar el flujo real de vehículos y analizar, por ejemplo, alternativas a implementar en los caminos mas transitados.

Serán consideradas y evaluadas dos estrategias:

1. *Ruteo estático: camino predeterminado y sin alternativas:* En esta estrategia, *c*ada auto antes de ser ingresado a la simulación es inicializado con el camino que seguirá para llegar a destino. El camino será determinado por algún algoritmo de ruteo en base al origen y destino del vehículo. El camino no variará durante la simulación. Esta estrategia, nos permite representar, por ejemplo, vehículos de transporte público los cuales siguen un trayecto fijo entre un determinado origen y destino.

2. *Ruteo dinámico: camino predetermino y con alternativas:* En este enfoque, al igual que en el anterior, cada auto es inicializado con el camino que seguirá para llegar a destino, antes de ingresar a la simulación. Pero, a diferencia de la alternativa anterior, los automóviles pueden tomar rutas alternativas durante el transcurso de la simulación, en caso de ocurrir ciertos incidentes, como por ejemplo al detectarse congestión. Esta estrategia, nos permite representar vehículos particulares, cuyos conductores inician un trayecto pensando en seguir un determinado camino, que puede variar por distintas causas como un embotellamiento o un accidente.

Monitoreo de congestión

Importancia del modelado de congestión

Un aspecto importante en la simulación de tráfico es la cantidad de tráfico existente en la región modelada. Este factor influirá no sólo en el desplazamiento de los vehículos sino también en el comportamiento y decisiones tomadas por el conductor del mismo.
En las grandes ciudades el rápido incremento de la cantidad de vehículos existentes (el cual no va acompañado con la adaptación de la estructura de calles, autopistas o caminos, que permita soportar este incremento) y la concentración de actividades en puntos específicos de la ciudad producen la formación de congestión.
La congestión siempre se localiza en determinados períodos de tiempo (por ejemplo, a la ida o vuelta de la jornada laboral) y sobre una región (por ejemplo, accesos a la ciudad desde zonas periféricas).
Como es evidente, que la congestión trae como consecuencia la disminución de la capacidad de movilidad de los vehículos, es muy importante detectar cuáles son los puntos conflictivos de manera de evaluar alternativas en la estructura de la ciudad (por ejemplo, el cambio de mano de alguna calle en determinada hora del día).
Por otra parte, es interesante tener en cuenta en el estudio del problema de congestión, las decisiones tomadas por los conductores al detectar la presencia de ella (y poder evaluar cómo varían dichas decisiones cuanta más información se posee), así como también conocer si hay caminos alternativos que permitan aliviar la congestión de un determinado sector.

Descripción general del modelo

En este trabajo se propone extender los modelos presentados anteriormente, incorporando una medida de congestión de tráfico, de manera poder tomar decisiones en base a esta información, pudiendo modificar el camino que los vehículos tenían planificados al inicio de la simulación.
Para ello, se realizaron los siguientes agregados y modificaciones:

- Se agregó un nuevo modelo DEVS denominado ‘controlador de congestión’.

Cada tramo poseerá un controlador de congestión asociado, el cual medirá la cantidad de autos que se encuentran dentro del mismo. Para ello, la primer y última celda del tramo se conectan con el controlador de congestión y le informan cada vez que ingresa o egresa un vehículo.

- Se modifican las reglas existentes en los cruces.

Ahora el cruce recibirá información de congestión del controlador de congestión a través de un port existente en las celdas de entrada del mismo.
En base a la disponibilidad de caminos para llegar a destino y la información de congestión del tramo al que se quiere ingresar, se decide seguir por dicho tramo o tomar por un camino alternativo.

Comportamiento del conductor

Un aspecto muy importante a considerar al analizar la dinámica del tráfico es el comportamiento del conductor, ya que es uno de los factores que caracterizan al movimiento de los vehículos.

Algunas de las características que se pueden considerar son las siguientes:

Velocidad deseada: Este aspecto va a influir sobre la velocidad a la que circularán los vehículos.

***Tiempo de reacción del conductor ante los eventos*.**
Como ejemplos se pueden mencionar:

- La disminución de velocidad en situaciones normales o de emergencia.
- La reacción ante una señal de tránsito, etc.

Impaciencia al seguir a un vehículo más lento: Influye en la decisión que tomará el conductor cuando se le presente la posibilidad de cambiar de carril.

Distancia crítica para cambiar de carril: Influye en la decisión que tomará el conductor cuando se le presente la posibilidad de cambiar de carril.

Distancia mínima de separación del auto que tiene delante: Tiene influencia en la forma que necesita el conductor para frenar el auto ante algún evento.

Todos estos aspectos afectarán a las siguientes características de los vehículos intervinientes en la simulación:
- Aceleración del vehículo, ya sea para aumentar su velocidad cuando tenga el camino libre, como para frenar ante diversos eventos. Esto influye, en consecuencia, en la velocidad a la que circulará el vehículo por la red.
- La decisión de cambiar de carril.
- El comportamiento al llegar a un cruce

Para incluir estas características en los modelos de simulación, se pueden definir tipos de conductores.
Para cada tipo se describen las características mencionadas anteriormente en distintas condiciones de tráfico.

Sólo a modo de ejemplo, presentamos la siguiente definición de los tipos de conductores y la matriz asociados a cada uno de ellos. Los valores que tomará la matriz dependerá de la definición de la función utilizada para calcular el movimiento que realizará el vehículo (velocidad a la que pasará).

Los tipos de conductores podrían ser:
- Prudente
- Normal
- Negligente

Los datos que caracterizan a cada tipo pueden ser:

	Libre	Normal	Congestionado
Velocidad deseada			
Tiempo de reacción ante los eventos			
Impaciencia para seguir a un vehículo lento			
Distancia para cambio de carril			
Distancia mínima de separación entre autos			
Respeto a señales de tráfico			

Cuando un auto ingresa a la simulación se le asigna el tipo de conductor (se guarda en el estado de la celda) y mediante una función que dado el tipo devuelve las características de mismo, se recuperan los datos necesarios para calcular el próximo movimiento que realizará el vehículo.

Los modelos han sido definidos de manera tal que, tanto los tipos de conductores, la matriz con sus características y la función que determina la reacción del vehículo, puedan ser redefinidas de

acuerdo a los distintos escenarios que deseen analizarse, sin tener que modificar las reglas que determinan el funcionamiento del mismo.

Por otra parte, también es importante tener en cuenta las características del vehículo, para considerar las posibilidades y limitaciones que ofrece el mismo.
Se puede tener en cuenta:

- La máxima velocidad que puede alcanzar.
- La velocidad normal a la que circula.
- La máxima aceleración / desaceleración que puede tomar (esta característica, depende por ejemplo, del tamaño del mismo).

Los modelos definidos también tienen en cuenta este aspecto, e incorporan el tipo de vehículo entre las características de los vehículos que participan de la simulación.

Sección 4.1: Ruteo estático - Camino predeterminado y sin alternativas

En este enfoque, al ingresar un auto al modelo se determina el camino que seguirá estáticamente. El mismo no cambiará durante la ejecución de la simulación.

Los caminos son construidos al nivel del lenguaje de especificación, es decir:

- Se determinan los tramos y los cruces con sus respectivos identificadores, con los cuales se puede construir un grafo que representaría la estructura del sector de ciudad a simular.

- Se construye una matriz origen – destino (OD) con los distintos pares origen – destino que resultaron del punto anterior.

- El origen y destino de los autos ingresados en el modelo son cruces.

- Para cada par origen – destino de la matriz OD se determina (mediante algún algoritmo de ruteo) el(los) mejor(es) camino(s). Puede obtenerse, además, para cada camino, información sobre el tiempo esperado de llegada a destino. Un camino consiste en una secuencia de tramos, que cumplen con las siguientes restricciones:

  Sea C (c_o, c_d) = $t_1.t_2……t_n$ un camino desde cruce c_o al cruce c_d:
  - El primer tramo se conecta al cruce de origen (siendo una de las salidas del mismo)
  - El último tramo se conecta al cruce de destino (siendo una de las entradas del mismo)
  - Para todo par t_i, t_{i+1} con $1 \leq i \leq n-1$, se cumple que t_i y t_{i+1} se encuentran unidos por un cruce c donde t_i es un tramo de entrada y t_{i+1} es un tramo de salida
  - El camino no contiene ciclos.

- Cada auto antes de ser incorpado al modelo es inicializado con el camino que tomará. El camino no variará durante la simulación. El camino será consumido durante la simulación, es decir cuando un auto ingresa a un tramo, este tramo es eliminado del camino, quedando el camino, en todo momento, formado por la secuencia de tramos que le falta recorrer al vehículo asociado al mismo (exceptuando el tramo actual).

Al utilizarse matrices ***pre-definidas*** que contendrán los caminos posibles para cada par origen destino, (matrices OD), se evita que los cálculos de caminos (algoritmos de ruteo) aumenten el tiempo requerido para la simulación.

Para poder aplicar esta estrategia, serán modificadas algunas especificaciones de los modelos con cruces definidos en [DW99].

Calles

Según lo definido en [DW99], una calle se especifica como una secuencia de *tramos*, cada uno de éstos representa un sector que se extiende a lo largo de una cuadra, donde todos los carriles tienen la misma dirección de circulación y una velocidad máxima permitida. Es decir, constituyen una sección de la calle sin cruces y de mano única. Consecuentemente para construir un sector *doble mano* es necesario definir un tramo en cada dirección.

En [DW99] un tramo fue especificado como:

Tramos = {(p1, p2, n, a, dir, max) / p1, p2 ∈ Puntos ∧ p1 ≠ p2 ∧ n, max ∈ N ∧ a,
dir ∈ {0, 1}}

donde,

- ***p1 y* p2:** representan los extremos del tramo y pertenecen al conjunto Puntos que se define como: Puntos = { (x,y) / x, y ∈ Z }. Se pide p1 ≠ p2, pues en el modelo no tiene sentido un tramo de longitud 0
- ***n ∈ N*:** indica la cantidad de carriles del tramo
- ***dir ∈ {0,1}*:** indica el sentido de circulación de los vehículos. Si dir = 1 los vehículos se desplazan hacia p2, caso contrario lo hacen hacia p1
- ***a ∈ {0, 1}*:** indica la forma del tramo. Aquí:
  - a = 0 ⇒ el tramo es un segmento recto determinado por los puntos p1 y p2.
  - a = 1 ⇒ el tramo queda definido por una de las semicircunferencias que se obtienen al partir la circunferencia cuyo diámetro es el segmento que une a p1 y p2. La circunferencia se parte utilizando el segmento que une a p1 y p2 como eje del corte, obteniendo 2 semicircunferencias con modelo subyacente idéntico y por ende no es necesario que la especificación diferencie ambos casos (este detalle quedará en el nivel de la interfaz con el modelador).
- ***max ∈ N*:** indica la velocidad máxima de circulación permitida en el tramo.

A esta especificación se agregó, a fin de incorporar la nueva funcionalidad de ruteo, una identificación unívoca del tramo: ***nro_tramo***, quedando la especificación del tramo como una tupla de siete elementos:

Tramos = {(p1, p2, n, a, dir, max, nro_tramo) / p1, p2 ∈ Puntos ∧ p1 ≠ p2 ∧ n, max ∈ N ∧ a, dir ∈ {0, 1}, nro_tramo ∈ N }

La necesidad de identificar unívocamente los tramos surge para determinar los caminos a seguir, deberá ser construido un grafo con los identificadores de cada tramo y cruce, que representaría la estructura del sector a simular. Sobre este grafo se aplica algún algoritmo de ruteo, para obtener el(los) mejor(es) camino(s) para cada par origen-destino.

En las siguientes secciones se definen los tramos asumiendo que se encuentran acoplados a cruces en los 2 extremos.

Modelo para calles de carril único

Definimos un tramo t = (p1, p2, 1, a, dir, max, nro_tramo) de carril único como un modelo **Cell-Devs** de una dimensión, cuya estructura se presenta en la siguiente figura.

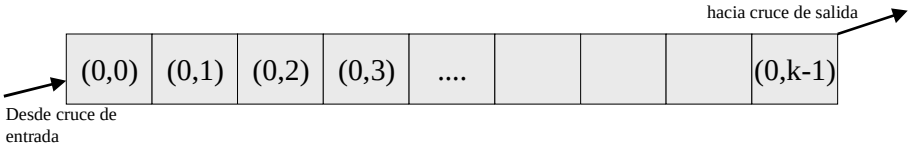


Figura 1 – Tramo de carril único

Cada celda en este espacio se define como:

$$C_{0j}\ (nro\_tramo) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = <η, P^x, P^y>, donde /*Interfaz del modelo */

η = 3
P^x = { (X_1, Record), (X_2, Record), (X_3, Record) }
P^y = { (Y_1, Record), (Y_2, Record), (Y_3, Record) }

$X, Y \in N$

A fin de incorporar la nueva funcionalidad de ruteo fue necesario agregar al estado de la celda el camino a seguir por el auto que se encuentra en la celda. En el ruteo estático el camino está predeterminado para cada auto antes de comenzar la simulación. El camino será representado como una secuencia de identificadores de tramos.

S : {s, phase, σqueue, σ}

donde s = (Hay_auto, camino)

$$Hay\_auto = \begin{cases} 1 \text{ si hay un vehículo en la celda} \\ 0 \text{ sino.} \end{cases}$$

$$Camino = \begin{cases} \{t_1.t_2.\ldots\ldots t_n\} \text{donde } ti \in N \wedge (\forall i\ (\exists\ r \in Tramos/\ Nro\_tramo(r) = ti\ )\ ) \\ 0 \text{ sino.} \end{cases}$$

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

(0,-1)	(0,0)	(0,1)

Figura 2 - Vecindario de la celda origen

delay = transport

d = Conversión_Demora(velocidad(max))

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-Devs para demoras de transporte

τ: S x N $\rightarrow$ S se define de la siguiente manera:

Función τ		
Nuevo estado	**Estado del vecindario**	**Nombre de la regla**
Hay_auto = 1 *Camino* = Camino (0,-1)	Hay_Auto (0,-1) = 1 and Hay_Auto(0,0) = 0	*Llega_Desde_Atrás*
Hay_auto = 0 *Camino* = 0	Hay_Auto(0,0) = 1 and Hay_Auto(0,1) = 0	*Sale_Hacia_Adelante*
(0,0)	t /*en otro caso conserva estado*/	*Default*

El estado de una celda representa la presencia o ausencia de un auto. Además, en el caso de que una celda este ocupada, la misma contiene el camino a seguir por el vehículo que se encuentra en la misma. Si la celda se halla vacía, el valor del camino es 0.
El movimiento de los vehículos queda definido por tres reglas:

- *Llega_Desde_Atrás:* representa el movimiento recto del vehículo desde la celda de atrás hacia la celda de origen, siempre y cuando esta última este vacía. Durante este movimiento el auto y el camino asociado se mueven hacia la celda de adelante.

- *Sale_Hacia_Adelante:* representa el movimiento recto del vehículo que se encuentra en la celda origen hacia la de adelante. Luego de realizar este movimiento la celda actual queda con los siguientes valores: hay_auto = 0 y camino = 0.

- *Default:* considera cualquier otro caso donde el estado de la celda no cambia.

Modelo acoplado

Un tramo se halla acoplado a un cruce en cada uno de sus extremos: celdas **(0,0)** y **(0,k-1).**

Al modelo acoplado definido en [DW99] se realizaron los siguientes cambios:

1. Se agregó la identificación unívoca del tramo.
2. Se agregaron ports que se utilizan para informar el camino desde los tramos a los cruces y viceversa.

El modelo acoplado para el tramo t = (p1, p2, 1, a, dir, max, nro_tramo) quedó definido como:

TC1(k, max, nro_tramo) = < Xlist, Ylist, I, X, Y, n, {t_1,...,t_n}, η, N, C, B, Z, select >

TC1(k, max, nro_tramo) significa Tramo de 1 Carril de longitud k (celdas) con velocidad máxima de circulación de max (Km/h) y cuya identificación es nro_tramo.

donde,

Ylist = { (0,0), (0,k-1) }

Xlist = { (0,0), (0,k-1) }

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,0), binario>, <$X_{\eta+1}$(0,k-1), binario>, **<$X_{\eta+2}$(0,0), Natural>**}
P^y = {<$Y_{\eta+1}$(0,0), binario>, <$Y_{\eta+1}$(0,k-1), binario>, **<$Y_{\eta+2}$(0,k-1), Natural>**}

Nota: Los ports resaltados en **negrita**, son los agregados para incorporar la funcionalidad de ruteo.

PORTS		
Port	**Nombre**	**Comentario**
Celda (0,0) : Primer celda del tramo		
$X_{\eta+1}$(0,0)	x-c-hayauto	Este port se utiliza para informar de la salida de un auto desde el cruce hacia el tramo (portvalue($X_{\eta+1}$) = 1).
$X_{\eta+2}$(0,0)	x-c-camino	Este port se utiliza para informar el camino del auto que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+2}$) = Camino).
$Y_{\eta+1}$(0,0)	y-c-haylugar	Este port se utiliza para que el cruce pueda saber si hay lugar en el tramo para que un auto pueda salir de él (portvalue($Y_{\eta+1}$) = 0).
Celda (0,k-1) : Ultima celda del tramo		
$X_{\eta+1}$(0,k-1)	x-c-haylugar	Este port permite saber si en el cruce hay lugar (portvalue($X_{\eta+1}$) = 0) para que avance un auto desde el tramo.
$Y_{\eta+1}$(0,k-1)	y-c-hayauto	Este port informa al cruce de la presencia de un auto en el tramo que desea avanzar hacia él (portvalue($Y_{\eta+1}$) = 1).
$Y_{\eta+2}$(0,k-1)	y-c-camino	Este port se utiliza para informar el camino del auto que sale del tramo para ingresar al cruce (portvalue($Y_{\eta+2}$) = Camino).

$\boldsymbol{X} \in \mathrm{N}$

$\boldsymbol{Y} \in N$
$\mathbf{n} = 1$
$\mathbf{t_1} = k$
$\eta = 3$
$\mathbf{N} = \{ (0,-1), (0,0), (0,1) \}$

$\mathbf{C} = \{ C_{ij} / i = 0 \wedge j \in [0, k-1] \}$, donde cada C_{ij} es un componente Cell-Devs con demora y la especificación de cada celda ha sido descripta antes de introducir el modelo acoplado.

$\mathbf{B} = \{ (0,0), (0,k-1) \}$

Z se construye siguiendo la definición dada por el formalismo Cell-Devs, instanciada con el vecindario de este espacio.

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

Especificación de la celdas borde

El comportamiento para las celdas borde es distinto al definido anteriormente. Para las celdas (0,0) y (0, k-1) se define un vecindario y comportamiento distinto al de las celdas internas del tramo. Entre otras cosas, las celdas borde deberán comunicarse con los cruces a través de ports.

Especificación de la celda (0,0)

Las características no mencionadas no varían con respecto a las otras celdas.

La celda (0,0) tiene el vecindario formado por ella misma y la celda de adelante. Además se encuentra acoplada a la celda del cruce desde la que recibe los vehículos.
Utilizando el port y-c-haylugar el cruce puede saber si hay lugar en el tramo para que un auto pueda salir del mismo. Para informar de la salida de un auto desde el cruce hacia el tramo se utiliza el port x-c-hayaauto.
Al ingresar un auto desde un cruce hacia un tramo, también se informará el camino a seguir por el mismo a través de un port (x-c-camino).
El camino a seguir por un auto se va consumiendo a medida que el auto avanza por los distintos tramos. Al ingresar un auto desde un cruce hacia un tramo, el camino se informa a través del port x-c-camino. Pero no es informado el camino completo, sino que al camino original se le elimina el próximo tramo que recorrerá el auto, es decir el tramo al cuál ingresará el auto desde el cruce. Formalmente:

Sea $c = t_1.t_2.\ldots\ldots t_n$ el camino original

Al ingresar el auto desde un cruce al tramo t_1 el camino informado al cruce será

$c' = t_2 \ldots\ldots t_n$.

De este modo cuando el auto está recorriendo el último tramo el camino será vacío.

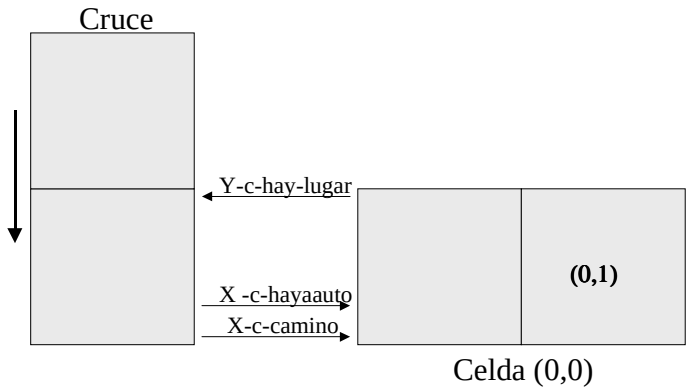


Figura 3- Vecindario y acoplamiento de la celda (0,0)

$\eta = 2$
N = { (0,0), (0,1) }

FUNCION τ			
Nuevo Estado	**Estado del vecindario**	**Nuevo estado de los ports**	**Nombre de la regla**
Hay_Auto =0 *Camino* = 0	Hay_Auto(0,0) = 1 and Hay_Auto(0,1) = 0	Send(0, y-c-haylugar)	*Sale_Hacia_Adelante*
Hay_Auto=1 *Camino* = portvalue(x-c-camino)	portvalue(x-c-hayauto) = 1 and Hay_Auto(0,0) = 0	Send(1, y-c-haylugar)	*Llega_Desde_Cruce*
(0,0)	t /*en otro caso conserva estado */		*Default*

Especificación de la celda (0,k-1)

Como puede observarse en la figura, la celda (0,k-1) tiene el vecindario formado por ella y la celda de atrás. Además se encuentra acoplada a una celda del cruce. De la misma manera en que fue definido en [DW99], para que un vehículo pueda ingresar al cruce deberá chequear que la celda a la que quiere ingresar y la anterior se encuentren vacías. Al ingresar un auto al cruce, se informará el camino a seguir por el mismo a través de un port (y-c-camino).

η = 2
N = { (0,-1), (0,0) }

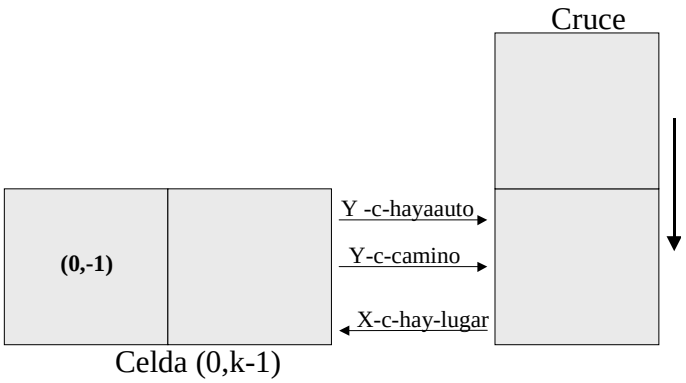


Figura 4 - Vecindario y acoplamiento de la celda (0,k-1)

FUNCION τ Y TIPO DE DEMORA				
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**	**Delay**	**Nombre de la regla**
Hay_Auto=0 *Camino*= 0	Hay_Auto(0,0) = 1 And portvalue(x-c-haylugar) = 0	Send(1, y-c-hayauto) Send (Camino(0,0), y-c-camino)	Inercial	*Sale_Hacia_Cruce*
Hay_Auto=1 *Camino*= Camino (0,-1)	Hay_Auto(0,-1) = 1 And Hay_Auto(0,0) = 0	Send(0, y-c-hayauto)	Transport	*Llega_Desde_Atrás*
(0,0)	t /*en otro caso conserva estado */	send(0, y-c-hayauto)	Transport	*Default*

Modelo para calles de más de un carril

Para poder aplicar esta estrategia en los modelos de más de dos carriles, deberán realizarse a los modelos definidos en [DW99], las mismas modificaciones que para el caso de un carril.

Como vimos en la sección anterior, se agrega a la especificación del tramo una identificación unívoca del tramo: ***nro_tramo.*** Esto es independiente del número de carriles.
Un tramo se define como

Tramos = {(p1, p2, n, a, dir, max, nro_tramo)

donde n representa la cantidad de carriles.

En el caso de tramos de más de un carril, los autos pueden realizar movimientos en diagonal, es decir pueden pasar de un carril a otro. Esto no afecta, en ningún caso, el ruteo de vehículos ya que al cambiar de carril, un auto sigue estando en el mismo tramo, es decir no varía su ruta. El ruteo de vehículos se realiza únicamente en los cruces.

Cruces

A la especificación de los cruces definida en [DW99] se agregó, a fin de incorporar la nueva funcionalidad de ruteo, una identificación unívoca del cruce: ***nro_cruce***

El conjunto de los cruces se obtiene a partir de los tramos definidos como:

$$
\begin{aligned}
&Cruces = \{ (c, maxc, nro\_cruce) / maxc \in \mathbb{N}, nro\_cruce \in \mathbb{N} \wedge \\
&\quad \exists t, t' \in Tramos \wedge t = (p1, p2, n, a, dir, max, nro\_tramo) \wedge \\
&\quad t' = (p1', p2', n', a', dir', max', nro\_tramo') \wedge \\
&\quad t \neq t' \\
&\quad \wedge (p1 = c \vee p2 = c) \\
&\quad \wedge (p1' = c \vee p2' = c) \\
&\}
\end{aligned}
$$

Los elementos de este conjunto se definen como puntos del espacio de dos dimensiones que representan los lugares donde se cruzan 2 ó más tramos. Un cruce siempre debe tener por lo menos un tramo de ingreso y uno de salida [DW99]. Esta restricción se escribe como:

$$
\begin{aligned}
&\forall (c, maxc) \in Cruces: \\
&\exists t, t' \in Tramos \wedge t = (p1, p2, n, a, dir, max, nro\_tramo) \wedge t' = (p1', p2', n', a', dir', max', \\
&nro\_tramo') \wedge t \neq t' \wedge [(p1 = p1' = c \wedge dir \neq dir') \vee (p1 = p2' = c \wedge dir = dir') \vee (p2 = p2' = c \wedge \\
&dir \neq dir')]
\end{aligned}
$$

Las intersecciones se representan como un anillo de celdas que se acopla con carriles de tramos, siguiendo la propuesta que fuera planteada en [CQL95], [CLQ96] y [CDL97]. Las reglas de comportamiento de los vehículos establecen que un auto dentro de la intersección (en el anillo) tiene prioridad para acceder a una posición sobre cualquier otro vehículo que esté fuera de ella. Dentro del cruce, un vehículo gira en sentido contrario a las agujas del reloj o sale, es decir no se permite que un auto se detenga a la espera de una salida particular porque esto puede provocar que en algún momento, nadie se pueda mover por estar esperando que otro vacíe la posición (deadlock). Cada vehículo avanza hacia una celda vacía, que puede estar dentro de la intersección o ser la primera de un carril de salida del cruce.

Un vehículo sale del cruce cuando llega al tramo que indica el camino predeterminado del auto (siempre que el mismo esté desocupado). El camino representa la secuencia de tramos que le falta recorrer a un vehículo, siendo el primer elemento del camino el próximo tramo al que deberá ingresar el auto. Si el identificador del próximo tramo que debe recorrer el auto es igual al identificador del tramo de salida, el auto abandona el cruce. En caso contrario, permanece en el mismo.

Si en el cruce se determina que el auto llegó a destino, el mismo es removido de la simulación

Cada cruce (c, maxc, nro_cruce) $\in$ Cruces, se define como un modelo *Cell-Devs* de una dimensión con demora de transporte y bordes conectados, cuya estructura se presenta en la figura.

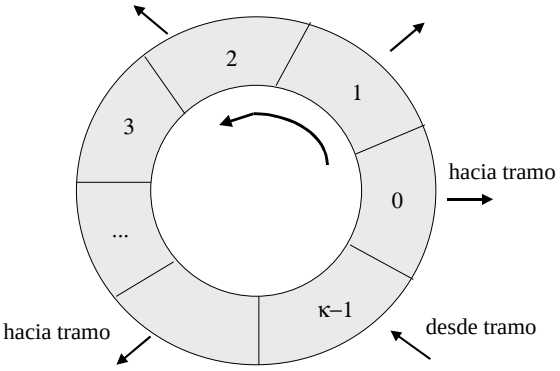


Figura 5- Cruce

Cada celda del espacio se define como:

$$C_{0j}\ (nro\_cruce) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = < η, P^X, P^Y>, donde

η = 3
P^X = { (X_1, Record), (X_2, Record), (X_3, Record) }
P^Y = { (Y_1, Record), (Y_2, Record), (Y_3, Record) }

$X, Y \in N$

Al fin de incorporar la nueva funcionalidad de ruteo fue necesario agregar al estado de la celda definido en [DW99], el camino a seguir por el auto que se encuentra en la celda, un identificador del cruce y el tramo al cuál se halla conectada la celda.

S: {s, phase, σqueue, σ}

donde s = (Hay_auto, nro_cruce, camino, nro_tramo)

Hay_auto = { 1 si hay un vehículo en la celda ; 0 sino.

nro_cruce ∈ N: identificador unívoco del cruce

Camino = { {$t_1.t_2$……t_n} donde ti ∈ N ∧ (∀i (∃ r ∈ Tramos/ Nro_tramo(r) = ti)) ; 0 sino.

Nro_tramo ∈ N: identificador del tramo al cuál se halla conectada la celda

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

delay = transport

d = Conversión_Demora(velocidad(maxc))
donde, maxc es la constante especificada para el cruce y representa la velocidad máxima de circulación permitida (en km/h).

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-Devs para demoras de transporte.

La definición de la función τ se realizará luego de introducir el modelo acoplado porque todas las celdas utilizan la información de los ports para calcular su nuevo estado.

El estado de una celda representa la presencia o ausencia de un vehículo, especificando el camino a seguir por el auto dentro de ella, un identificador único del cruce y el tramo al cuál se halla conectada la misma. Dentro del cruce sólo se permite que los vehículos avancen hacia la posición de adelante, siempre y cuando la misma esté vacía. De esta forma el vecindario necesario para establecer el nuevo estado de una celda está constituido por ella misma, la celda anterior y la siguiente.
El cruce es el encargado de rutear los vehículos que llegan a él. Por esta razón se incluyó en el estado de la celda el tramo al cual se hallan conectados. De este modo, el cruce rutea los vehículos por el tramo correspondiente, según el camino que deben seguir.

Modelo acoplado

El modelo acoplado correspondiente al cruce (c, maxc) se define como:

Cruce(k, In, Out, maxc) = < Xlist, Ylist, I, X, Y, n, {t_1,...,t_n}, η, N, C, B, Z, select >

donde,

Ylist = { (0,i) / $0 \leq i < k$}

Xlist = { (0,i) / $0 \leq i < k$}

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,i), binario>, <$X_{\eta+2}$(0,i), binario>**, <$X_{\eta+3}$(0,i), Natural>/ $0 \leq i < k$** }
P^y = {<$Y_{\eta+1}$(0,i), binario>, <$Y_{\eta+2}$(0,i), binario>, **<$Y_{\eta+3}$(0,i), Natural> / $0 \leq i < k$** }

Nota: Los ports resaltados en **negrita**, son los agregados para incorporar la funcionalidad de ruteo.

PORTS		
Port	**Nombre**	**Comentario**
Celdas de entrada del cruce		
$X_{\eta+1}$(0,i), i ∈ In	x-t-hayauto	Este port se usa para saber si en el tramo existe un auto que desea ingresar al cruce (portvalue(x-t-hayauto) = 1).
$X_{\eta+3}$(0,i), i ∈ In	x-t-Camino	Este port se usa para conocer el camino del auto que ingresa al cruce
$Y_{\eta+1}$(0,i), i ∈ In	y-t-haylugar	Este port informa al tramo si hay suficiente lugar en el cruce para el avance un auto.
Celdas de salida del cruce		
$X_{\eta+2}$(0,i), i ∈ Out	x-t-haylugar	Este port se usa para saber si en el tramo existe lugar para que salga un auto del cruce (portvalue(x-t-haylugar) = 0).
$Y_{\eta+2}$(0,i), i ∈ Out	y-t-hayauto	Este port informa al tramo si hay un auto que sale desde el cruce hacia él.
$Y_{\eta+3}$(0,i), i ∈ Out	y-t-camino	Este port informa al tramo el camino del auto que sale desde el cruce hacia él.

X ∈ N

Y ∈ N

n = 1

$\mathbf{t_1}$ = k

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

C = { C_{ij} / i = 0 $\wedge$ j $\in$ [0, k-1] }, donde cada C_{ij} es un componente Cell-Devs con demora.

B = {$\varnothing$}

Z se construye siguiendo la definición dada por el formalismo Cell-Devs, instanciada con el vecindario de este espacio.

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

Cruce(k, In, Out, maxc) significa que es un Cruce de longitud k (celdas) con velocidad máxima de circulación permitida de maxc (Km/h), donde las posiciones de In actúan como entradas hacia el cruce y las de Out son salidas del mismo. Los conjuntos In y Out se obtienen utilizando la función Ports_In_Out,

$$\{I, O\} = \text{Ports\_In\_Out}((c, maxc), Tramos)$$

Para establecer cuáles son las celdas de entrada y cuáles las de salida, se parte de la especificación de los tramos que lo tienen como extremo y se chequea el sentido de circulación de los vehículos en ellos. En el Apéndice E se muestra la función Ports_In_Out que realiza dicha tarea y devuelve dos conjuntos, el primero contiene cada tramo de ingreso al cruce que está conectado al mismo con el índice de la primera celda del cruce con la que se acopla; y el otro contiene la misma información pero para los tramos de salida. Esta función establece un ordenamiento de los tramos que se acoplan al mismo cruce, de forma tal que los tramos más cercanos entre sí ocupen posiciones vecinas, esto se logra ordenándolos según el ángulo de inclinación respecto a la recta y = y1, donde c = (x1,y1). Este orden también es necesario para realizar el acoplamiento de los modelos.
De los conjuntos I y O sólo se necesitan los índices de las celdas que son entradas y las que son salidas. Esto es:

$$In = \{ i+j / \exists t = (c1, c2, n, a, dir, max) \wedge (t,i) \in I \wedge j \in [0, n-1] \}$$

$$Out = \{ i+j / \exists t = (c1, c2, n, a, dir, max) \wedge (t,i) \in O \wedge j \in [0, n-1] \}$$

Para determinar la cantidad de celdas que tiene el cruce (c, maxc), basta sumar el número de carriles que tiene cada tramo asociado al cruce pues cada celda se acopla a un carril. Es decir,

$$k = \sum_{\substack{(t,i)\in(I\cup O)\wedge \\ t=(c1,c2,n,a,dir,max)}} n \qquad = \#(In) + \#(Out)$$

La especificación de este modelo representa el movimiento de los vehículos dentro de las intersecciones de calles. Cada celda del cruce se interconecta con algún tramo, por lo tanto la interfaz del modelo queda conformada por todas ellas. Los ports definidos tienen como finalidad informar sobre el estado de la celda del cruce a la del tramo y viceversa, esto se utiliza para poder calcular su nuevo estado.

Para definir la función τ se debe tener en cuenta que el comportamiento de las celdas difiere si se trata de celdas de ingreso o de salida del cruce, por lo tanto se define por separado para cada caso. El comportamiento de las celas varían con respecto a las definidas en [DW99], ya que se debe incorporar a las mismas el ruteo de vehículos.

Celdas de ingreso al cruce

Las celdas de ingreso al cruce son:

{ (0,i) / i $\in$ In }

FUNCION τ		
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Hay_auto = 1 *Nro_cruce*=nro_cruce(0,0) *Nro_tramo*=nro_Tramo(0,0) *Camino*=Camino(0,-1)	Hay_Auto(0,0) = 0 and Hay_Auto(0,-1) = 1 */* Ingresa a la celda un auto que ya estaba dentro del cruce */*	send(1, y-t-haylugar)
Hay_auto = 1 *Nro_cruce*=nro_cruce(0,0) *Nro_tramo*= Nro_tramo(0,0) *Camino*= portvalue(x-t-Camino)	Hay_Auto(0,0) = 0 and portvalue(x-t-hayauto) = 1 and ¬Es_Destino(portvalue(x-t-camino)) */* Entra al cruce un auto que todavía no ha llegado a destino */*	send(1, y-t-haylugar)
Hay_auto = 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Camino*= 0	Hay_Auto(0,0) = 0 and portvalue(x-t-hayauto) = 1 and Es_Destino(portvalue(x-t-camino)) */* Elimina el auto del modelo ya que ha llegado a destino */*	send(0, y-t-haylugar)
Hay_auto = 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Camino*= 0	Hay_Auto(0,0) = 1 and Hay_Auto(0,1) = 0 and Hay_Auto(0,-1) = 0 */* No hay un auto con prioridad dentro del cruce */*	send(0, y-t-haylugar)
Hay_auto = 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= 0 *Camino*= 0	Hay_Auto(0,0) = 1 and Hay_Auto(0,1) = 0 and Hay_Auto(0,-1) = 1 */* Hay un auto con prioridad dentro del cruce */*	send(1, y-t-haylugar)
(0,0)	T */* En otro caso la celda conserva el estado */*	-

El valor que puede recibir una celda de ingreso al cruce por los ports externos es 1 si existe un vehículo que avanza desde el tramo (recibiendo en este caso también el camino que seguirá el vehículo), 0 en otro caso. El valor que envían estas celdas hacia los tramos es 1 si no hay suficiente espacio (2 celdas vacías) para que pueda ingresar un vehículo al cruce, y 0 en caso contrario. Esto significa que por el port y-t-haylugar se envía 0 sólo cuando la celda origen y la anterior están vacías, permitiendo que un auto ingrese desde el tramo sólo cuando no hay otro dentro del cruce que quiera ocupar la misma posición.

Las celdas de ingreso al cruce pueden recibir vehículos desde su vecina de atrás o desde el tramo acoplado (port x-t-hayauto).

El comportamiento de las celdas de entrada se define con seis reglas

- La primera regla modela el ingreso a la celda de origen de un vehículo que ya se encontraba dentro del cruce. Para ello la celda de origen debe estar vacía y la vecina de atrás debe estar ocupada. En esta regla se actualiza el port y-t-haylugar con 1 indicando al tramo que la celda está ocupada.

- La segunda regla modela el ingreso a la celda de origen de un vehículo que aún no ha llegado a destino desde el tramo acoplado. Esto último es determinado por la función Es_destino que

recibe como parámetro el camino asociado al vehículo. La función Es_destino retorna verdadero si el camino es vacío y falso en otro caso. Si el camino es vacío, indica que no existen tramos por recorrer, es decir que el vehículo ha llegado a destino.

- La tercera regla elimina el vehículo que ha ingresado al cruce ya que el mismo llegó al destino (la función Es_Destino retorna True).

- La cuarta y quinta regla representan que la celda origen se vacía si la de adelante está libre; diferenciándose en el estado de la celda de atrás. Si esta última está vacía entonces se actualiza el port con valor 0 pues hay suficiente lugar para que ingrese un vehículo desde el tramo, caso contrario (Hay_Auto(0,-1) = 1) el estado del port se actualiza con 1 porque hay un auto dentro del cruce que tiene prioridad de acceder a la celda origen y los coches del tramo no pueden entrar en la intersección.

- La sexta regla representa que la celda conserva su estado en cualquier otro caso no contemplado en las condiciones anteriores.

Celdas de salida del cruce

Las celdas de salida del cruce son:

$\{ (0,i) / i \in Out \}$

	FUNCION τ	
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Hay_auto = 1 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Camino*= Camino(0,-1)	Hay_Auto(0,0) = 0 and Hay_Auto(0,-1) = 1 and ((portvalue(x-t-haylugar) = 1) or (portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) = nro_tramo(0,0))) /* Llega auto que permanecerá dentro del cruce */	send(0, y-t-hayauto)
Hay_auto = 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Camino*= 0	Hay_Auto(0,0) = 0 and Hay_Auto(0,-1) = 1 and portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) = nro_tramo(0,0) /* Llega auto que abandonará el cruce */	send(1, y-t-hayauto) send(ColaCamino(Camino(0,-1)), y-t-camino)
Hay_auto = 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Camino*= 0	Hay_Auto(0,0) = 1 and Hay_Auto(0,1) = 0	send(0, y-t-hayauto)
(0,0)	t /*En otro caso la celda conserva el estado */	send(0, y-t-hayauto)

El valor que recibe una celda de salida del cruce por los ports externos es el estado de la celda del tramo acoplada (0 si está vacía, 1 si hay un auto). Cuando un vehículo se encuentra en un celda de salida del cruce, tiene 2 opciones para su próximo movimiento: puede salir hacia el tramo

acoplado o no salir (avanza dentro del cruce). Esta elección se efectúa cuando el auto ingresa en la celda origen y se refleja en su nuevo estado, que es 1 si permanece en el cruce y 0 si sale hacia el tramo. También la actualización del port externo (y-t-hayauto) es diferente, toma valor 1 si decide salir y 0 en otro caso.
Si el próximo tramo que el vehículo deberá recorrer es igual al identificador del tramo que se halla acoplado a la celda origen, el vehículo abandonará el cruce. En caso contrario, permanecerá dentro del mismo.
La Función Proximo_Tramo(Camino) devuelve el primer tramo del camino pasado como parámetro.

El comportamiento de las celdas de salida se define con 4 reglas:

- La primera regla representa el avance de un vehículo hacia la celda origen, modelando el caso en que el auto permanecerá dentro del cruce. Esta elección se debe a que la celda del tramo acoplado está ocupada (portvalue(x-t-haylugar) = 1) o el tramo acoplado no corresponde al próximo tramo que deberá recorrer el auto según el camino aosciado al mismo. Como el auto no saldrá hacia el tramo, la celda actualiza con 0 el port y-t-hayauto.

- La segunda regla representa el avance de un vehículo hacia la celda origen, modelando el caso en que el auto saldrá del cruce hacia el tramo acoplado a la celda. Para que esto sea posible la celda del tramo acoplado debe estar vacía y además el tramo debe estar indicado en el camino a seguir. La celda actualiza con 1 el port y-t-hayauto para que el tramo detecte la salida del vehículo y pasa a través del port y-t-camino la ruta del vehículo.

- La tercera regla representa que el vehículo de la celda de origen avanza hacia la celda siguiente dentro del cruce; siempre y cuando esta última se encuentre vacía. Cabe destacar que en este caso la celda origen tenía estado 1, que representa que el vehículo debe avanzar hacia la celda (0,1) en su próximo movimiento (es decir, en este caso no puede salir del cruce).

- Por último, la cuarta regla representa que la celda conserva su estado en cualquier otro caso no contemplado en las condiciones anteriores.

Acoplamiento entre Cruces y Tramos

El acoplamiento de estos modelos representa el movimiento de vehículos que ingresan al (salen del) cruce desde (hacia) algún tramo. Un cruce c = (p, maxc, nro_cruce) tiene influencias sobre los tramos t que lo tienen como un extremo, es decir, el conjunto de modelos que influye el Cell-Devs de c es el siguiente:

$I_c = \{ M_t / t \in \text{Tramos} \wedge t = (p1, p2, n, a, dir, max, nro\_tramo) \wedge (p1 = p \text{ OR } p2 = p) \}$

Notación

M_t representa al modelo DEVS o Cell-Devs definido para t, donde t es un elemento del lenguaje de especificación (por ejemplo, i puede ser un tramo, cruce, semáforo, etc.).

Luego un tramo t tiene definidas las influencias sobre los modelos de los dos cruces que tienen como extremo, es decir

$$I_t = \{M_{c1}\} \cup \{M_{c2}\}, \text{ si } t = (p1, p2, n, a, dir, max, nro\_tramo) \text{ y } (\exists v1, v2 \in \mathbb{N}: c1, c2 \in \text{Cruces} \wedge c1 = (p1, v1) \wedge c2 = (p2, v2))$$

Para completar la definición del acoplamiento entre cruces y tramos hay que establecer a través de qué ports se comunican. Al definir el modelo Cell-DEVS para un cruce, se agrega una celda por cada carril de cada tramo que lo tiene como extremo; a través de ella se producirá el ingreso (la salida) del vehículo al (del) cruce. Por lo tanto, cada celda del borde (columnas 0 y k-1, si el tramo tiene k celdas) de cada tramo tiene una celda del cruce asociada, con la que debe intercambiar la información sobre su estado.

Ports de salida tramo – ingreso cruce

- $Y_{\eta+1}$ (y-c-hayauto) (Tramo) → $X_{\eta+1}$ (x-t-hayauto) (Cruce) Informando la salida de un auto desde el tramo hacia el cruce (celdas de ingreso al cruce).
- $Y_{\eta+2}$ (y-c-camino) (Tramo) → $X_{\eta+3}$ (x-t-camino) (Cruce) Informando el camino del vehículo (celdas de ingreso al cruce).
- $Y_{\eta+1}$ (y-t-haylugar) (Cruce) → $X_{\eta+1}$ (x-c-haylugar) (Tramo) Informando si hay lugar en el cruce para que pueda salir un auto del tramo.

Ports de salida cruce – ingreso tramo

- $Y_{\eta+2}$ (y-t-hayauto) (Cruce) → $X_{\eta+1}$ (x-c-hayauto) (Tramo) Informando la salida de un auto desde el cruce hacia el tramo
- $Y_{\eta+3}$ (y-t-camino) (Cruce) → $X_{\eta+2}$ (x-c-camino) (Tramo) Informando el camino del vehículo
- $Y_{\eta+1}$ (y-c-haylugar) (Tramo) → $X_{\eta+2}$ (x-t-haylugar) (Cruce) Informando si hay lugar en el tramo para que pueda ingresar un auto desde el cruce.

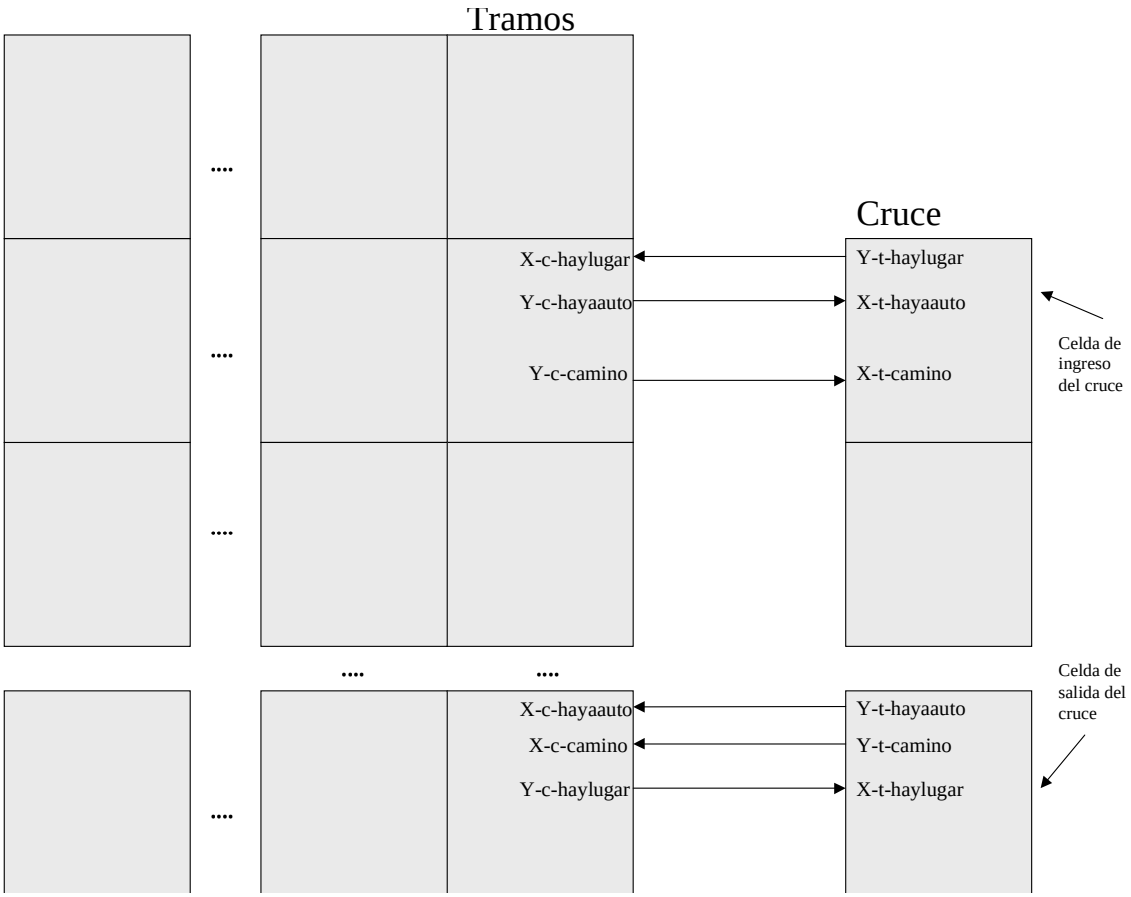


Figura 6 - Ports de acoplamiento entre tramos y cruces

Los ports se acoplan definiendo la función Z. La definición de Z es:

Para cada $(t,i) \in I$ con t = (p1, p2, n, a, dir, max, nro_tramo)

$$Z_{tc} : Y_{\eta+1}(j, k-1)_t \rightarrow X_{\eta+1}(0,i+j)_c,$$
$$Y_{\eta+2}(j, k-1)_t \rightarrow X_{\eta+3}(0,i+j)_c \ \forall \ (j \in N, j \in [0, n-1])$$
$$Z_{ct} : Y_{\eta+1}(0,i+j)_c \rightarrow X_{\eta+1}(j, k-1)_t, \ \forall \ (j \in N, j \in [0, n-1])$$

Para cada $(t,i) \in O$ con t = (p1, p2, n, a, dir, max, nro_tramo)

$$Z_{ct} : Y_{\eta+2}(0,j+i)_c \rightarrow X_{\eta+1}(n-1-j, 0)_t,$$
$$Y_{\eta+3}(0,j+i)_c \rightarrow X_{\eta+2}(n-1-j, 0)_t, \ \forall \ (j \in N, j \in [0, n-1])$$
$$Z_{tc} : Y_{\eta+1}(n-1-j, 0)_t \rightarrow X_{\eta+2}(0,j+i)_{c,} \ \forall \ (j \in N, j \in [0, n-1])$$

Sección 4.2: Monitoreo de congestión y Ruteo dinámico

MONITOREO DE CONGESTION

DESCRIPCIÓN GENERAL DEL MODELO

En este trabajo se propone extender los modelos presentados anteriormente, incorporando una medida de congestión de tráfico, de manera de poder tomar decisiones en base a esta información, pudiendo modificar el camino que los vehículos tenían planificados al inicio de la simulación.
Para ello, se realizaron los siguientes agregados y modificaciones:

- Se agregó un nuevo modelo DEVS denominado 'controlador de congestión'.

Cada tramo poseerá un controlador de congestión asociado, el cual medirá la cantidad de autos que se encuentran dentro del mismo. Para ello, la primer y última celda del tramo se conectan con el controlador de congestión y le informan cada vez que ingresa o egresa un vehículo.

- Se modifican las reglas existentes en los cruces.

Ahora el cruce recibirá información de congestión del controlador de congestión a través de un port existente en las celdas de entrada del mismo.
En base a la disponibilidad de caminos para llegar a destino y la información de congestión del tramo al que se quiere ingresar, se decide seguir por dicho tramo o tomar por un camino alternativo.

Controladores de Congestión

El objetivo de estos modelos es medir la cantidad de tráfico existente en cada una de las calles.
Cómo se ha explicado anteriormente, cada controlador de congestión monitorea un tramo, por lo tanto, se encuentra acoplado al mismo.
Como los tramos pueden poseer uno o varios carriles, la definición del modelo DEVS que representa al controlador de congestión puede variar de acuerdo a esta característica.

- Si el tramo contiene un sólo carril: el controlador de congestión recibe información desde la primer y última celda del tramo. Por lo tanto, posee dos ports de entrada.
  En su estado almacena la cantidad de autos que se encuentran en el tramo. Al recibir un mensaje desde la primer celda incrementa en 1 su estado, mientras que si proviene desde la última se decrementa el estado en la misma cantidad.
  Posee un sólo port de salida que informa la cantidad de autos existentes en el tramo que monitorea.

- Si el tramo contiene n carriles: el controlador de congestión recibe información desde las primeras (celdas (i,0) con $0 \leq i \leq n-1$) y últimas (celdas (i,k-1) con $0 \leq i \leq n-1$, donde k es la cantidad de celdas del tramo) celdas del tramo.
  Se presentan dos alternativas posibles con respecto a este modelo, que varían en la forma que se almacena la información recibida y que posibilitan distintos niveles de análisis de la misma.
  Las alternativas son las siguientes:
  1. La más simple: En el estado del controlador se guarda un número natural, que representa la cantidad de autos existentes en todo el tramo. En este caso, el controlador de congestión se define exactamente igual que para el caso de un carril. La diferencia radica en el acoplamiento con el tramo, ya que todas las celdas celdas (i,0) con $0 \leq i \leq n-1$ se conectarán con el port x-r-hayautoin y todas las celdas (i,k-1) con $0 \leq i \leq n-1$ se conectarán con el port x-r-hayautoout.
     Posee un sólo port de salida que informa la cantidad de autos existentes en el tramo que monitorea.
  2. Más sofisticada: En el estado del controlador se guarda un vector con una posición por cada carril. Si llamamos a este vector 'Cant_Autos', Cant_Autos(i) representa la cantidad de autos que contiene el carril i.
     Por lo tanto, para un tramo de n carriles, el controlador de congestión poseerá 2n ports de entrada (n para registrar la entrada de vehículos al tramo y n para registrar su salida). Denominamos $x\text{-}r\text{-}hayautoin_i$ a los ports de entrada que se conectan a las celdas de entrada al tramo y $x\text{-}r\text{-}hayautoout_i$ a los que se conectan a las de salida.
     La llegada de un mensaje por el port $x\text{-}r\text{-}hayautoin_i$, incrementa en 1 la posición i del vector. La llegada de un mensaje por el port $x\text{-}r\text{-}hayautoout_i$, decrementa en 1 la posición i del vector.
     Esta opción tiene sentido si la función 'Hay_Congestión', utilizada para decidir el ruteo del vehículo y presentada más adelante, es lo suficientemente inteligente (y eficiente) para sacar provecho de esta información.
     Posee un sólo port de salida que informa la cantidad de autos existentes por cada carril del tramo que monitorea.

La primer y segunda alternativa para controladores de congestión de tramos de varios carriles, se diferencian en los siguientes casos:

Supongamos casos tales que, sólo uno (o algunos) de los carriles se encuentra congestionado, esto puede darse cuando sólo un determinado carril está habilitado para realizar una maniobra (por ejemplo, girar), o el tramo posee carriles especiales (por ejemplo, para transporte público), etc.
En estas situaciones, la distribución de tráfico no va a ser uniforme en todos los carriles y dependiendo de las características del vehículo (camino elegido, tipo de vehículo, etc), le es últil conocer dicha distribución.
A continuación se presentan dos figuras que ilustran estas situaciones.

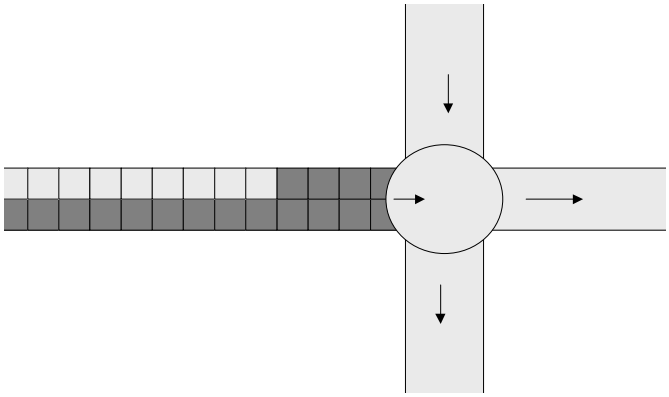


Figura 1 - Ejemplo de distribución de tráfico

Esta figura puede representar que:
- Sólo el carril derecho se encuentra habilitado para doblar.
- Lindero a ese tramo se encuentra un teatro, cine o shopping y existen muchos autos estacionados en 'doble fila'.

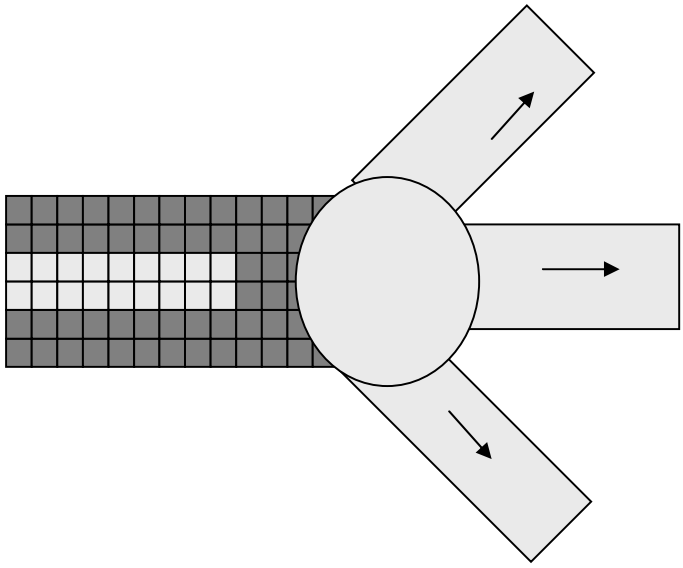


Figura 2 - Ejemplo de distribución de tráfico

Esta figura puede representar que:
- Sólo los carriles 'externos' se encuentran habilitados para doblar (a tramos distintos) y los carriles centrales se usan para seguir por el tramo que se encuentra en línea recta (asumamos que el cambio de carril es dificultoso en tan poco espacio).

El controlador de congestión que reconoce el tráfico por carriles puede detectar estas situaciones.

Definición para controladores que monitorean tramos de un carril:

Un controlador de congestión es un modelo atómico.
Definición:

$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$

Donde:
$\boldsymbol{X}$ = {$<X_{1,}$ Natural>, $<X_2$, Natural>}

$\boldsymbol{Y}$ = {$<Y_{1,}$ Natural>}

Los ports se denominarán de la siguiente forma:

PORTS		
Port	**Nombre**	**Comentario**
X_1	x-r-hayautoin	Este port se utiliza para informar de la entrada de un auto al tramo que es controlado por este modelo. (portvalue(X_1) = 1).
X_2	x-r-hayautoout	Este port se utiliza para informar de la salida de un auto del tramo que es controlado por este modelo. (portvalue(X_2) = 1).
Y_1	y-r-peso	Este informa el peso (portvalue(Y_1) =S)

S = k $\in$ N done k representa la cantidad de autos que contiene en un momento dado el tramo que controla este modelo.

δ_{int} = Id (Función identidad ya que no se producen cambios de estado por transiciones internas)

δ_{ext}
Cuando recibo 1 en el port x-r-hayautoin
Se suma 1 al estado (s' = s+1, siendo s' el nuevo estado)
Pasivar

Cuando recibo 1 en el port x-r-hayautoout
Se resta 1 al estado (s' = s-1, siendo s' el nuevo estado)
Pasivar

λ define como:
Esta función devuelve el valor del estado (s) por el port y-r-peso

Ta = 0

Definición para controladores que monitorean tramos de varios carriles:

Primer alternativa: el controlador almacena en su estado la cantidad de autos existentes en todo el tramo.

La definición es la misma que para el caso de un carril.

Segunda alternativa: el controlador almacena en su estado la cantidad de autos existentes en cada carril del tramo.

Un controlador de congestión es un modelo atómico.
Definición:

$M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$

Donde:
$\boldsymbol{X}$ = {<X_i, Natural> con $0 \leq i \leq 2n-1$} donde n es la cantidad de carriles del tramo que monitorea

$\boldsymbol{Y}$ = {<Y_1, Natural>}
Si asumimos que cada carril del tramo no puede poseer más de 99 vehículos (con dos dígitos alcanza) se define el port de salida de la siguiente forma:
Sea $Y_1 = a_n \ldots\ldots a_0 \in N$, para el carril i con $0 \leq i \leq n-1$, el peso del tramo es el nro $a_{2i+1}a_{2i}$

Los ports se denominarán de la siguiente forma:

PORTS		
Port	**Nombre**	**Comentario**
X_i Con $0 \leq i \leq n-1$	x-r-hayautoin$_i$	Estos ports se utilizan para informar de la entrada de un auto al tramo que es controlado por este modelo. (portvalue(X_i) = 1).
X_i Con $n \leq i \leq 2n-1$	x-r-hayautoout$_i$	Estos ports se utilizan para informar de la salida de un auto del tramo que es controlado por este modelo. (portvalue(X_i) = 1).
Y_1	y-r-peso	Este informa el peso (portvalue(Y_1) =S)

S = (V_0, ……, V_{n-1}) donde $V_i \in N$ con $0 \leq i \leq n-1$
V_i representa la cantidad de autos que contiene en un momento dado el carril i del tramo que controla este modelo.

$\boldsymbol{\delta_{int}}$ = Id (Función identidad ya que no se producen cambios de estado por transiciones internas)

$\boldsymbol{\delta_{ext}}$
Cuando recibo 1 por cualquier port x-r-hayautoin$_i$
Se suma 1 a la posición correspondiente al carril i
(siendo s' el nuevo estado donde ($\forall q \neq i : V_q' = V_q$) and $V_i' = V_i + 1$)
Pasivar
Cuando recibo 1 por cualquier port x-r-hayautoout$_i$
Se resta 1 a la posición correspondiente al carril i
(siendo s' el nuevo estado donde ($\forall q \neq i : V_q' = V_q$) and $V_i' = V_i - 1$)
Pasivar

$\boldsymbol{\lambda}$ define como:
Esta función devuelve el valor del estado (s) por el port y-r-peso

Ta = 0

Definición de los modelos para incorporar el monitoreo de congestión

Calles

Definimos un tramo t = (p1, p2, 1, a, dir, max, nro_tramo) de carril único como un modelo **Cell-Devs** de una dimensión, cuya estructura se presenta en la siguiente figura.

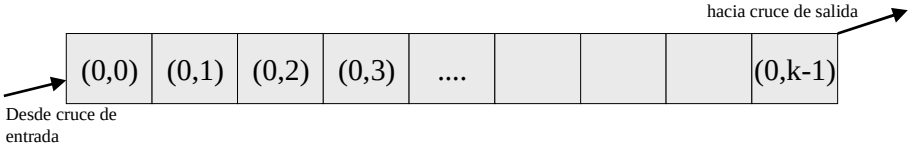


Figura 3 – Tramo de carril único

Cada celda en este espacio se define como:

$$C_{0j}\ (nro\_tramo) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = <η, P^x, P^y>, donde /*Interfaz del modelo */

η = 3

P^x = { (X_1, Record), (X_2, Record), (X_3, Record) }

P^y = { (Y_1, Record), (Y_2, Record), (Y_3, Record) }

$X, Y \in N$

S : {s, phase, σqueue, σ}

donde s = (Destino, camino)

$$Destino \in N = \begin{cases} \neq 0 \text{ si hay un vehículo en la celda} \\ 0 \text{ sino.} \end{cases}$$

$$Camino = \begin{cases} \{t_1.t_2 \ldots\ldots t_n\} \text{ donde } ti \in N \wedge (\forall i\ (\exists\ r \in Tramos/\ Nro\_tramo(r) = ti\ )\ ) \\ 0 \text{ sino.} \end{cases}$$

Notas:

- A diferencia de la sección anterior donde se guardaba en el estado la presencia o ausencia de un vehículo en la celda (Hay_Auto), ahora se almacena en el estado el destino del vehículo. Este cambio, se debe a que ahora nos interesa conocer explícitamente adonde se dirije el vehículo para poder obtener un camino alternativo en caso de ser necesario. Si el atributo 'Destino' contiene el valor 0, esto indica que la celda se encuentra vacía.
- Si no hay auto en la celda tampoco hay camino

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

(0,-1)	(0,0)	(0,1)

Figura 4 - Vecindario de la celda origen

delay = transport

d = Conversión_Demora(velocidad(max))

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-Devs para demoras de transporte

τ: S x N $\rightarrow$ S se define de la siguiente manera:

Función τ		
Nuevo estado	**Estado del vecindario**	**Nombre de la regla**
Destino = Destino(0,-1) *Camino* = Camino(0,-1)	Destino(0,-1) != 0 And Destino(0,0) = 0	*Llega_Desde_Atrás*
Destino = 0 *Camino* = 0	Destino(0,0) != 0 And Destino(0,1) = 0	*Sale_Hacia_Adelante*
(0,0)	t /*en otro caso conserva estado*/	*Default*

Esta especificación no varía respecto al modelo de ruteo presentado en la sección 4.1
El movimiento de vehículo queda definido por tres reglas:

- *Llega_Desde_Atrás:* Ingresa a la celda el auto que se encontraba en la celda anterior.
- *Sale_Hacia_Adelante:* Avanza el vehículo a la próxima celda.
- *Default:* Considera cualquier otro caso donde el estado de la celda no cambia.

Modelo acoplado

Un tramo se halla acoplado a un cruce en cada uno de sus extremos: celdas **(0,0)** y **(0,k-1).**
Al modelo acoplado definido en [DW99] se realizaron los siguientes cambios:

1. Se agregó la identificación unívoca del tramo, necesaria para ser utilizada por el algoritmo de ruteo.
2. Se agregaron ports que se utilizan para informar el camino desde los tramos a los cruces y viceversa.
3. Se agregaron los ports que informan al modelo de congestión la entrada y salida de vehículos del tramo, de manera que pueda calcular cuántos autos se encuentran dentro de él.

El modelo acoplado para el tramo t = (p1, p2, 1, a, dir, max, nro_tramo) quedó definido como:

TC1(k, max, nro_tramo) = < Xlist, Ylist, I, X, Y, n, {t_1,...,t_n}, η, N, C, B, Z, select >

TC1(k, max, nro_tramo) significa Tramo de 1 Carril de longitud k (celdas) con velocidad máxima de circulación de max (Km/h) y cuya identificación es nro_tramo.

donde,

Ylist = { (0,0), (0,k-1) }

Xlist = { (0,0), (0,k-1) }

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,0), Natural>,
<$X_{\eta+2}$(0,0), Natural>,
<$X_{\eta+1}$ (0,k-1), Binario>, }
P^y = {<$Y_{\eta+1}$ (0,0), Binario>,
<$Y_{\eta+2}$(0,0), binario>,
<$Y_{\eta+1}$(0,k-1), Natural>,
<$Y_{\eta+2}$(0,k-1), binario>,
<$Y_{\eta+3}$ (0,k-1), Natural>}

Nota: Los ports resaltados en **negrita**, son los agregados para monitorear congestión

PORTS		
Port	**Nombre**	**Comentario**
Celda (0,0) : Primer celda del tramo		
$X_{\eta+1}(0,0)$	x-c-destino	Este port se utiliza para informar de la salida de un auto desde el cruce hacia el tramo, informando su destino (portvalue($X_{\eta+1}$) = Destino).
$X_{\eta+2}$ (0,0)	x-c-camino	Este port se utiliza para informar el camino del auto que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+2}$) = Camino).
$Y_{\eta+1}$ (0,0)	y-c-haylugar	Este port se utiliza para que el cruce pueda saber si hay lugar en el tramo para que un auto pueda salir de él (portvalue($Y_{\eta+1}$) = 0).
$Y_{\eta+2}$ (0,0)	y-r-EntraAuto	Este port informa al controlador de congestión la entrada de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
Celda (0,k-1) : Ultima celda del tramo		
$X_{\eta+1}$ (0,k-1)	x-c-haylugar	Este port permite saber si en el cruce hay lugar (portvalue($X_{\eta+1}$) = 0) para que avance un auto desde el tramo.
$Y_{\eta+1}$ (0,k-1)	y-c-destino	Este port informa al cruce de la presencia de un auto en el tramo que desea avanzar hacia él, informando su destino (portvalue($Y_{\eta+1}$) = Destino).
$Y_{\eta+2}$ (0,k-1)	y-r-SaleAuto	Este port informa al controlador de congestion la salida de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+3}$ (0,k-1)	y-c-camino	Este port se utiliza para informar el camino del auto que sale del tramo para ingresar al cruce (portvalue($Y_{\eta+3}$) = Camino).

$\boldsymbol{X} \in N$
$\boldsymbol{Y} \in N$
$\mathbf{n} = 1$
$\mathbf{t_1} = k$
$\boldsymbol{\eta} = 3$
$\mathbf{N} = \{ (0,-1), (0,0), (0,1) \}$

$\mathbf{C} = \{ C_{ij} / i = 0 \wedge j \in [0, k-1] \}$, donde cada C_{ij} es un componente Cell-Devs con demora.

$\mathbf{B} = \{ (0,0), (0,k-1) \}$

Z se construye siguiendo la definición dada por el formalismo Cell-Devs, instanciada con el vecindario de este espacio.

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

Especificación de la celdas borde

El comportamiento para las celdas borde es distinto al definido anteriormente. Para las celdas (0,0) y (0, k-1) se define un vecindario y comportamiento distinto al de las celdas internas del tramo. Ahora, las celdas borde deberán comunicarle al controlador de congestión, la entrada y salida de vehículos del tramo.

Especificación de la celda (0,0)

Las características no mencionadas no varían con respecto a las otras celdas.

El comportamiento de esta celda es similar al descripto en los modelos anteriores, la única diferencia radica en su comunicación con el nuevo modelo de control de congestión. Cada vez que un auto ingresa al tramo, él mismo debe avisarle al controlador de congestión para que actualice el valor correspondiente a la cantidad de autos dentro de él.
Por lo tanto, al aplicarse la regla 'Llega_Desde_Cruce' (que produce el ingreso del auto), se envía a través del port conectado al controlador el aviso.

$$\eta = 2$$

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

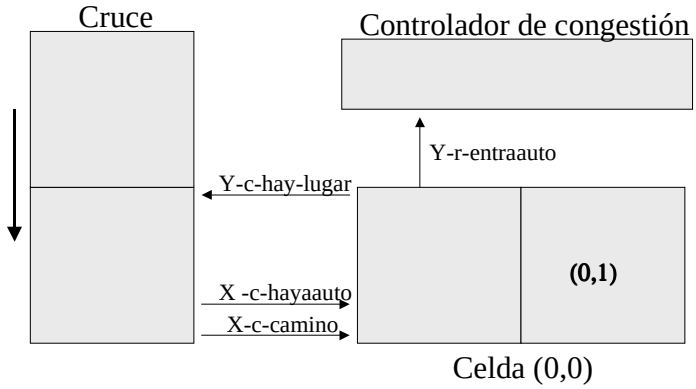


Figura 5 - Vecindario y acoplamiento de la celda (0,0)

FUNCION τ			
Nuevo Estado	**Estado del vecindario**	**Nuevo estado de los**	**Nombre de la regla**

		ports	
Destino = 0 *Camino* = 0	Destino(0,0) != 0 And Destino(0,1) = 0	Send(0, y-c-haylugar)	*Sale_Hacia_Adelante*
Destino = portvalue(x-c-destino) *Camino* = portvalue(x-c-camino)	Portvalue(x-c-destino) != 0 And Destino(0,0) = 0	Send(1, y-r-EntraAuto) Send(1, y-c-haylugar)	*Llega_Desde_Cruce*
(0,0)	t /*en otro caso conserva estado*/		*Default*

Los demás parámetros del modelo para esta celda no cambian.

Especificación de la celda (0,k-1)

El controlador de congestión necesita conocer el evento de salida de un auto del tramo, de manera de descontar dicho auto de la cantidad de autos contenidos en el tramo.
Para ello, se agrega a esta celda un nuevo port de salida que la acopla al controlador de congestión, informándole a través de él, la salida de cada auto.
Cada vez que se ejecuta la regla 'Sale_Hacia_Cruce', la cual produce la salida del auto, se le envía al cruce esta información a través del port 'y-r-saleauto'.

$\eta = 2$

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

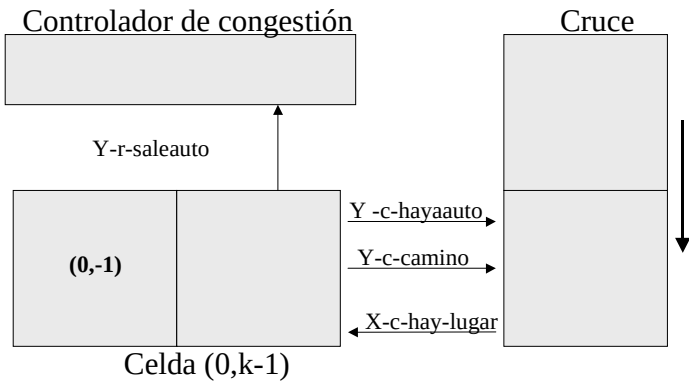


Figura 6 - Vecindario y acoplamiento de la celda (0,k-1)

FUNCION τ Y TIPO DE DEMORA				
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**	**Delay**	**Nombre de la regla**

Destino = 0 *Camino* = 0	Destino(0,0) != 0 and portvalue(x-c-haylugar) = 0	/* Al cruce */ Send(Destino(0,0), y-c-destino) Send (Camino(0,0), y-c-camino) /* Al controlador de congestión */ Send(1, y-c-SaleAuto)	Inercial	*Sale_Hacia_Cruce*
Destino = Destino(0,-1) *Camino*= Camino (0,-1)	Destino (0,-1) != 0 And Destino (0,0) = 0	Send(0, y-c-destino)	Transport	*Llega_Desde_Atrás*
(0,0)	t /*en otro caso conserva estado */	Send(0, y-c-destino)	Transport	*Default*

Los demás parámetros del modelo para esta celda no cambian.

Calles con varios carriles

Para el caso de calles con varios carriles, es necesario modificar el modelo acoplado y las reglas de los modelos que provocan el ingreso y egreso de autos al tramo.
La especificación completa de los modelos de calles de varios carriles puede encontrarse en [DW99]. Aquí sólo se presentará la modificación necesaria para informar al controlador de congestión, la entrada y salida de vehículos.

Así como para el caso de un carril se agregó un port de salida tanto de la primer como de la última celda, a través de los cuales se informa al controlador de congestión el ingreso y egreso de vehículos, se generaliza este concepto para los tramos de varios carriles.
Para un tramo de n carriles:

- En cada celda del tramo (i,0), donde $0 \le i \le n-1$, se agrega un port de salida y-r-EntraAuto$_i$ que se conectará al port de entrada del controlador de congestión por el cual se notifica de la entrada de vehículos al tramo.
- En cada celda del tramo (i,k-1), donde $0 \le i \le n-1$, se agrega un port de salida y-r-SaleAuto$_i$ que se conectará al port de entrada del controlador de congestión por el cual se notifica de la salida de vehículos al tramo.

Esquema de acoplamiento del tramo con la alternativa más simple del controlador de congestión:

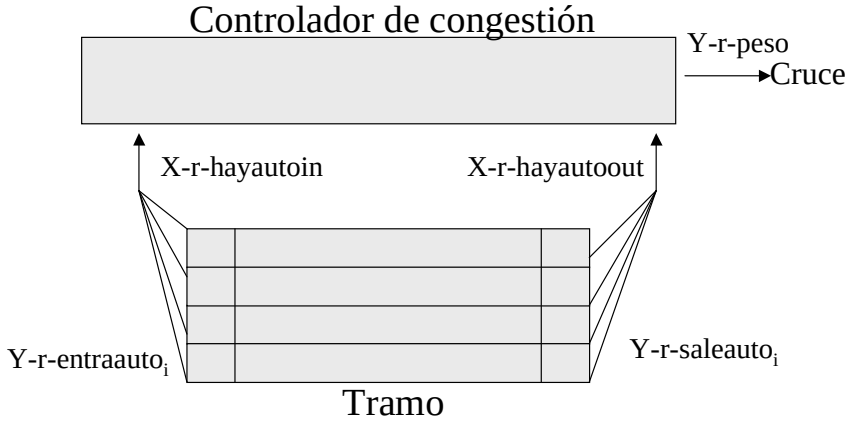
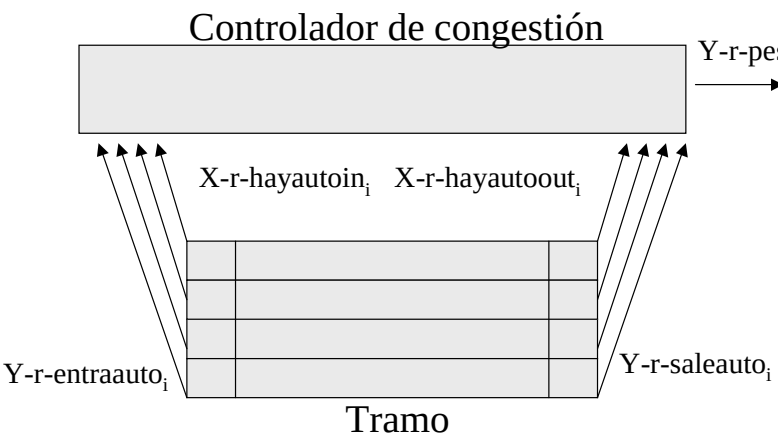


Figura 7 - Acoplamiento entre un tramo de varios carriles y el controlador de congestión

Esquema de acoplamiento del tramo con la alternativa más 'sofisticada' del controlador de congestión:



El mo ... **ramo) se define** ... **T** ... **Z, select >**

TC(k, ... cada uno, con velocid ... o.

donde, ...

P^x = {<$X_{\eta+1}$(i,0), Natural> / 0 ≤ i < #c} U
{<$X_{\eta+1}$(i,k-1), binario> / 0 ≤ i < #c} U
{<$X_{\eta+2}$(i,0), Natural> / 0 ≤ i < #c}

P^y = {<$Y_{\eta+1}$(i,0), binario> / 0 ≤ i < #c} U
{<$Y_{\eta+1}$(i,k-1), Natural> / 0 ≤ i < #c} U
{<$Y_{\eta+2}$(i,0), binario> / 0 ≤ i < #c} U
{<$Y_{\eta+2}$(i,k-1), Binario> / 0 ≤ i < #c} U
{<$Y_{\eta+3}$(i,k-1), Natural> / 0 ≤ i < #c}

Nota: Los ports resaltados en **negrita**, son los agregados para monitorear congestión

Figura 8 - Acoplamiento entre un tramo de varios carriles y el controlador de congestión

Modelo acoplado

Un tramo de n carriles se halla acoplado a un cruce en cada uno de sus extremos: celdas **(i,0)** y **(i,k-1)** con 0 ≤ i ≤ n-1.

PORTS		
Port	**Nombre**	**Comentario**

Celdas (i,0) con $0 \le i \le \#c-1$: Primer celda de cada carril del tramo		
$X_{\eta+1}(i,0)$, $0 \le i \le \#c-1$	x-c-destino$_i$	Estos ports se utilizan para informar de la salida de un auto desde el cruce hacia el tramo, informando su destino (portvalue($X_{\eta+1}$) = Destino).
$X_{\eta+2}$ (i,0) $0 \le i \le \#c-1$	x-c-camino$_i$	Estos ports se utilizan para informar el camino del auto que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+2}$) = Camino).
$Y_{\eta+1}(i,0)$, $0 \le i \le \#c-1$	y-c-haylugar$_i$	Estos ports se utilizan para que el cruce pueda saber si hay lugar en el tramo para que un auto pueda salir de él (portvalue($Y_{\eta+1}$) = 0).
$Y_{\eta+2}$ (i,0) $0 \le i \le \#c-1$	y-r-EntraAuto$_i$	Estos ports informan al controlador de congestion la entrada de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
Celda (i,k-1) con $0 \le i \le \#c-1$: Ultima celda de cada carril del tramo		
$X_{\eta+1}$(i,k-1), $0 \le i \le \#c-1$	x-c-haylugar$_i$	Estos ports permiten saber si en el cruce hay lugar (portvalue($X_{\eta+1}$) = 0) para que avance un auto desde el tramo.
$Y_{\eta+1}$(i,k-1), $0 \le i \le \#c-1$	y-c-destino$_i$	Estos ports informan al cruce de la presencia de un auto en el tramo que desea avanzar hacia él, informando su destino (portvalue($Y_{\eta+1}$) = Destino).
$Y_{\eta+2}$ (i,k-1) $0 \le i \le \#c-1$	y-r-SaleAuto$_i$	Estos ports informan al controlador de congestion la salida de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+3}$ (i,k-1) $0 \le i \le \#c-1$	y-c-camino$_i$	Estos ports se utilizan para informar el camino del auto que sale del tramo para ingresar al cruce (portvalue($Y_{\eta+3}$) = Camino).

$\boldsymbol{X} \in N$

$\boldsymbol{Y} \in N$

$\mathbf{n} = 2$

$\mathbf{t_1} = \#c$

$\mathbf{t_2} = k$

N y $\boldsymbol{\eta}$: su definición depende del modelo de celda de cada carril. Los mismos han sido presentados en [DW99]

C = { C_{ij} / $i \in [0, \#c-1] \wedge j \in [0, k-1]$ }, donde cada C_{ij} es un componente Cell-DEVS con demora.

B = { (i, 0) / $0 \le i \le \#c-1$ } U { (i, k-1) / $0 \le i \le \#c-1$ }

Z se construye siguiendo la definición dada por el formalismo Cell_DEVS, instanciada con el vecindario de este espacio.

La definición de la función **select** depende de la cantidad de carriles del tramo y se encuentra especificada en [DW99].

Para completar la modificación necesaria para el monitoreo de congestión, sobre la definición de modelos para tramos de varios carriles, se debe agregar a las reglas que representan la entrada y salida de vehículos, el envío de información al controlador de congestión.

Denominación de la regla	Modificación
Llega_desde_cruce	Send(1,y-r-EntraAuto$_i$)
Sale_hacia_cruce	Send(1,y-r-SaleAuto$_i$)

INCORPORACIÓN DE RUTEO DINÁMICO

Cruces

En esta sección se presenta la definición del modelo que representa a los cruces.
Los cruces serán los encargados de decidir si el auto conserva el camino que tenía planeado o intenta tomar uno alternativo, en función de la información de congestión que recibe.

En esta sección se presentan dos modelos Cell-Devs.
El primero de ellos, ha sido especificado de acuerdo a la definición conceptual de los modelos Cell-Devs. Al intentar implementar el mismo en la herramienta N-CD++, fue necesario realizar cambios debido a que la herramienta utiliza conceptos distintos en relación a la disponibilidad de información en los ports de los modelos (para una descripción más detallada, ver apéndice C).
Por este motivo, se ha especificado el segundo modelo, el cual posee la misma idea que el anterior pero se ajusta al concepto usado en N-CD++.

Descripción del comportamiento del cruce:
Cuando un vehículo llega al cruce, ingresa en la celda de entrada (con los valores recibidos por los ports conectados al tramo), ésta celda es la encargada de decidir el ruteo del auto, de acuerdo a la información de congestión recibida del controlador de congestión.
Si el camino que tiene el auto se encuentra congestionado le pedirá a la matriz OD un camino alternativo, y si existe, tomará el mismo. Luego hará avanzar el auto hacia la próxima celda del cruce.
No se evaluará el estado del camino alternativo (siendo éste un riesgo a correr en caso de decidir tomar el mismo).
Las celdas de entrada rutean sólo cuando el vehículo ingresa desde un tramo, no cuando llega a ella un auto que ya estaba dentro del cruce.

Definición

Cada celda del espacio se define como:

$$C_{0j}(nro\_cruce) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = $< \eta, P^X, P^Y >$, donde

$\eta = 3$
$P^X = \{ (X_1, \text{Record}), (X_2, \text{Record}), (X_3, \text{Record}) \}$
$P^Y = \{ (Y_1, \text{Record}), (Y_2, \text{Record}), (Y_3, \text{Record}) \}$

$X, Y \in N$

S: {s, phase, σqueue, σ}

donde s = (Destino, camino, nro_cruce, nro_tramo)

Destino = { !=0 si hay un vehículo en la celda, indica el destino del mismo
0 sino.

Camino = { $\{t_1.t_2.\ldots\ldots t_n\}$ donde $ti \in N \wedge (\forall i\ (\exists\ r \in Tramos/\ Nro\_tramo(r) = ti\ ))$
0 sino.

nro_cruce $\in$ N: identificador unívoco del cruce

nro_tramo $\in$ N: identificador del tramo que se conecta a la celda de salida del cruce

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

delay = transport

d = Conversión_Demora(velocidad(maxc))
donde, maxc es la constante especificada para el cruce y representa la velocidad máxima de circulación permitida (en km/h).

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

La definición de la función τ se realizará luego de introducir el modelo acoplado porque todas las celdas utilizan la información de los ports para calcular su nuevo estado.

El estado de una celda representa la presencia o ausencia de un vehículo, especificando el destino del mismo, el camino a seguir por el auto dentro de ella, un identificador único del cruce y el identificador del tramo al que se conecta cada celda de salida del cruce. Dentro del cruce sólo se permite que los vehículos avancen hacia la posición de adelante, siempre y cuando la misma esté vacía. De esta forma el vecindario necesario para establecer el nuevo estado de una celda está constituido por ella misma, la celda anterior y la siguiente.

Modelo acoplado

El modelo acoplado correspondiente al cruce (c, maxc) se define como:

Cruce(k, In, Out, maxc) = < Xlist, Ylist, I, X, Y, n, {t_1,...,t_n}, η, N, C, B, Z, select >

donde,

Ylist = { (0,i) / $0 \leq i < k$}

Xlist = { (0,i) / $0 \leq i < k$}

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,i), Natural>,
<$X_{\eta+2}$(0,i), Natural>,
<$X_{\eta+3}$(0,i), Natural> con $0 \leq i < k$ }

P^y = {<$Y_{\eta+1}$(0,i), Natural>,
<$Y_{\eta+2}$(0,i), Natural>,
<$Y_{\eta+3}$(0,i), Natural> con $0 \leq i < k$ }

Celdas de entrada

De acuerdo a lo descripto anteriormente, cada tramo tendrá acoplado un controlador del congestión. Llamemos $Congestión_j$ al controlador de congestión que monitorea al $Tramo_j$.
Al cruce le interesa conocer cuál es el estado de cada uno de los tramos de salida del mismo, de manera de poder evaluar si el tramo por el cual quiere salir el vehículo se encuentra congestionado.
Cómo la decisión de ruteo la toma cada celda de entrada del cruce, cada una de estas celdas se acoplan a todos los controladores de congestión que corresponden a tramos de salida del cruce.
Por lo tanto, para un cruce con l_s celdas de salida, en cada celda de entrada va a poseer l_s ports de entrada, cada uno de ellos perteneciendo a un controlador de congestión distinto.

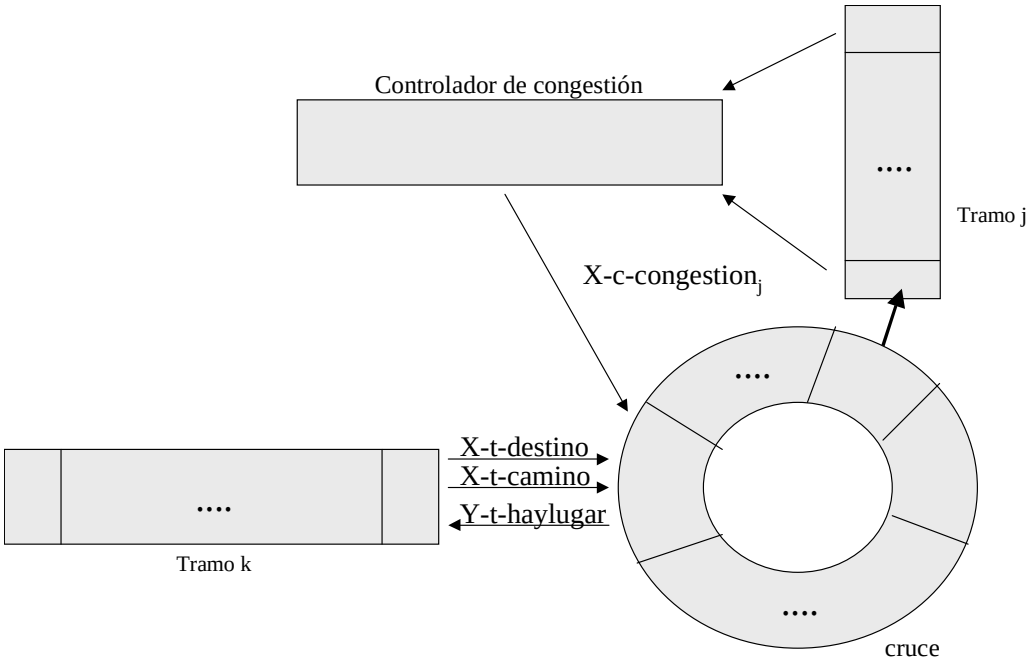


Figura 9 - Acoplamiento de una celda de entrada

PORTS		
Port	**Nombre**	**Comentario**
$X_{\eta+1}(0,i)$, $i \in$ In	x-t-destino	Este port se usa para saber si en el tramo existe un auto que desea ingresar al cruce (portvalue(x-t-destino) != 0), indicando el destino del mismo.
$X_{\eta+2}(0,i)$, $i \in$ In	x-t-Camino	Este port se usa para conocer el camino del auto que ingresa al cruce
$X_{\eta+3+j}(0,i)$, $i \in$ In Donde j es el nro de tramo al que le corresponde el crontrolador de congestión	x-c-congestion$_j$ Es decir, cada celda puede recibir información de todos los controladores de congestión que corresponden a los tramos de salida del cruce.	Por este port se conoce la información de congestión del tramo por el cual quiere salir el auto que está en el cruce.
$Y_{\eta+1}(0,i)$, $i \in$ In	y-t-haylugar	Este port informa al tramo si hay suficiente lugar en el cruce para el avance un auto.

Celdas de salida

Los ports que corresponden a las celdas de salida no varían respecto al modelo correspondiente al ruteo estático.

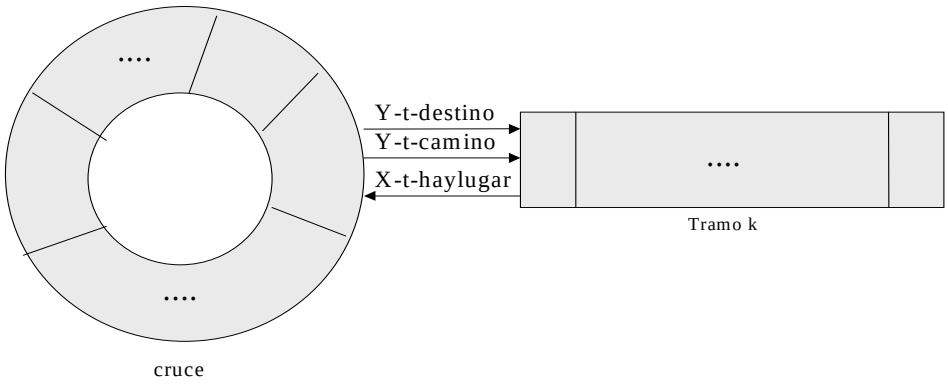


Figura 10 - Acoplamiento de una celda de salida del cruce

PORTS		
Port	**Nombre**	**Comentario**
$X_{\eta+1}(0,i)$, $i \in$ Out	x-t-haylugar	Este port se usa para saber si en el tramo existe lugar para que salga un auto del cruce (portvalue(x-t-haylugar) = 0).
$Y_{\eta+1}(0,i)$, $i \in$ Out	y-t-destino	Este port informa al tramo si hay un auto que sale desde el cruce hacia él, indicando el destino.
$Y_{\eta+2}(0,i)$, $i \in$ Out	y-t-camino	Este port informa al tramo el camino del auto que sale desde el cruce hacia él.

X $\in$ N
Y $\in$ N
n = 1
$\mathbf{t_1}$ = k
N = { (0,-1), (0,0), (0,1) }
C = { C_{ij} / i = 0 $\wedge$ j $\in$ [0, k-1] }, donde cada C_{ij} es un componente Cell-DEVS con demora y la especificación de cada celda ha sido descripta antes de introducir el modelo acoplado.
B = {$\emptyset$}
Z se construye siguiendo la definición dada por el formalismo Cell_DEVS, instanciada con el vecindario de este espacio.
select = { (0,1), (0,0), (0,-1) }

Cruce(k, In, Out, maxc) significa que es un Cruce de longitud k (celdas) con velocidad máxima de circulación permitida de maxc (Km/h), donde las posiciones de In actúan como entradas hacia el cruce y las de Out son salidas del mismo. Los conjuntos In y Out se obtienen utilizando la función Ports_In_Out,

{I, O} = Ports_In_Out((c, maxc), Tramos)

Ya descripta en secciones anteriores y cuya definición se encuentra en el apéndice E.

La especificación de este modelo representa el movimiento de los vehículos dentro de las intersecciones de calles. Cada celda del cruce se interconecta con algún tramo, por lo tanto la interfaz del modelo queda conformada por todas ellas.

Para definir la función τ se debe tener en cuenta que el comportamiento de las celdas difiere si se trata de celdas de ingreso o de salida del cruce, por lo tanto se define por separado para cada caso.

Celdas de ingreso al cruce

Las celdas de ingreso al cruce son:

$\{ (0,i) / i \in In \}$

FUNCION τ		
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = nro_Tramo(0,0)	Destino(0,0) = 0 and Destino(0,-1) != 0 */* Ingresa a la celda un auto que ya estaba dentro del cruce, conservando el mismo camino */*	Send(1, y-t-haylugar)
Destino = portvalue(x-t-destino) *Camino* = portvalue(x-t-Camino) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0)	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and portvalue(x-t-destino) != nro_cruce(0,0) and ¬Hay_Congestion(portvalue(x-c-congestion $_{Proximo\_Tramo(Camino(0,-1))}$) , nro_Tramo(0,0))) */* Entra al cruce un auto que todavía no ha llegado a destino, conservando el mismo camino con el que venía */*	Send(1, y-t-haylugar)
Destino = portvalue(x-t-destino) *Camino* = Nuevo_Camino(Nro_Cruce(0,0), Destino(0,-1), Camino(0,-1)) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0)	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and portvalue(x-t-destino) != nro_cruce(0,0) and Hay_Congestion(portvalue(x-c-congestion $_{Proximo\_Tramo(Camino(0,-1))}$) , nro_Tramo(0,0))) */* Entra al cruce un auto que todavía no ha llegado a destino, cambiando el camino por encontrarse con congestión */*	Send(1, y-t-haylugar)
Destino = 0 *Camino*= 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0)	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and portvalue(x-t-destino) = nro_cruce(0,0) */* Elimina el auto del modelo ya que ha llegado a destino */*	Send(0, y-t-haylugar)
Destino = 0 *Camino* = 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0)	Destino(0,0) != 0 and Destino(0,1) = 0 and Destino(0,-1) = 0 */* No hay un auto con prioridad dentro del	Send(0, y-t-haylugar)

	*cruce */*	
Destino = 0 *Camino* = 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0)	Destino(0,0) != 0 and Destino(0,1) = 0 and Destino(0,-1) != 0 */* Hay un auto con prioridad dentro del cruce */*	Send(1, y-t-haylugar)
(0,0)	T */* En otro caso la celda conserva el estado */*	-

Los valores que puede recibir una celda de ingreso al cruce por los ports externos indican:
- El destino y el camino, si existe un vehículo que avanza desde el tramo, 0 en otro caso.
- La información de congestión de los tramos de salida.

Los valores que envían estas celdas son:
- Hacia el tramo: es 1 si no hay suficiente espacio (2 celdas vacías) para que pueda ingresar un vehículo al cruce, y 0 en caso contrario. Esto significa que por el port y-t-haylugar se envía 0 sólo cuando la celda origen y la anterior están vacías, permitiendo que un auto ingrese desde el tramo sólo cuando no hay otro dentro del cruce que quiera ocupar la misma posición.

Las celdas de ingreso al cruce pueden recibir vehículos desde su vecina de atrás o desde el tramo acoplado (port x-t-destino, port x-t-camino).
Si hay un vehículo en su vecina de atrás y la celda está vacía, el auto avanza conservando el camino. Se le indica al tramo de entrada que el cruce se encuentra ocupado (Primer regla).
Si ingresa un vehículo desde el tramo acoplado pueden darse dos casos:
- Si el vehículo no llegó a destino, avanza el auto a la celda con los datos enviados desde el tramo acoplado y decide si conservará el camino que tenía planificado el vehículo. Se indica al tramo de entrada que el cruce se encuentra ocupado (segunda y tercer regla).
- Si el vehículo llegó a destino, se elimina el auto de la simulación, indicando al tramo de entrada que el cruce se encuentra libre (cuarta regla).

Si la celda de entrada ya contenía un auto y hay lugar en la celda de adelante, el mismo avanza. Las últimas dos reglas modelan este caso, diferenciándose en el estado de la celda de atrás. Si esta última está vacía entonces se actualiza el port que le indica al tramo si se puede entrar al cruce con valor 0 pues hay suficiente lugar para que ingrese un vehículo desde el tramo, caso contrario el estado del port se actualiza con 1 porque hay un auto dentro del cruce que tiene prioridad de acceder a la celda origen y los coches del tramo no pueden entrar en la intersección.

Celda de salida del cruce

Las celdas de salida del cruce son:

$$\{ (0,i) / i \in Out \}$$

	FUNCION τ	
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Destino = Destino(0,-1) *Camino*= Camino(0,-1) *nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0)	Destino(0,0) = 0 and Destino(0,-1) != 0 and ((portvalue(x-t-haylugar) = 1) or (portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) != Nro_Tramo(0,0)))	Send(0, y-t-destino)

	/* Llega auto que permanecerá dentro del cruce */	
Destino = 0 *Camino*= 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0)	Destino(0,0) = 0 and Destino(0,-1) != 0 and Portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) = Nro_Tramo(0,0) /* Llega auto que abandonará el cruce */	Send(Destino(0,-1), y-t-destino) Send(Cola_Camino(Camino(0,-1)), y-t-camino)
Destino = 0 *Camino*= 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= 0	Destino(0,0) != 0 and Destino(0,1) = 0 /* Sigue en el cruce */	Send(0, y-t-destino)
(0,0)	t /*En otro caso la celda conserva el estado */	Send(0, y-t-destino)

El comportamiento de las celdas de salida es el siguiente:
Cuando un vehículo llega a una celda de salida del cruce, tiene 2 opciones para su próximo movimiento: puede salir hacia el tramo acoplado o no salir (avanza dentro del cruce).
Esta decisión se basa en los siguientes factores:
1. El tramo acoplado lo conduce a destino?
2. Hay lugar para ingresar al mismo?

En base a las respuestas a estas preguntas varía la decisión a tomar (es decir, la regla que se ejecuta).
El auto permanece en el cruce cuando (primer regla):
- No hay lugar en el tramo acoplado al cruce, o
- Si bien hay lugar en el tramo, no corresponde salir por el mismo.

En caso contrario, sale del cruce, siguiendo su camino por el tramo acoplado (segunda regla). En este caso, le envía al tramo acoplado el camino que le queda por recorrer al vehículo (es decir, va consumiendo el camino una vez que se ingresa al tramo indicado en el mismo)

La tercer regla indica que el vehículo de la celda de origen avanza hacia la celda siguiente dentro del cruce; siempre y cuando esta última se encuentre vacía. Cabe destacar que en este caso la celda origen tenía un vehículo dentro de ella, que representa que el vehículo debe avanzar hacia la celda (0,1) en su próximo movimiento (es decir, en este caso no puede salir del cruce).

Para definir las reglas anteriormente explicadas se han utilizado las siguientes funciones:
- Función Nuevo_Camino(Origen, Destino, CaminoActual)
  Dado el cruce de partida (origen), el cruce al cual se quiere llegar (Destino) y el camino actual que sigue el auto, devuelve un camino alternativo que conduzca del origen al destino. La elección del camino alternativo puede definirse dentro de la función, dando mayor flexibilidad a este modelo.

- Función Proximo_Tramo(Camino)
  Devuelve el primer tramo del camino pasado como parámetro.

- Función Cola_Camino(Camino)
  Devuelve el mismo camino pasado como parámetro, pero sin el primer tramo. Se usa para ir consumiendo el camimo a medida que se transita por los tramos.

- Función Hay_Congestion(Valor_congestión, Identificación del tramo)
  Devuelve un valor booleano que indica si el tramo está muy congestionado. Una posible definición puede ser que un x% del tramo se encuentre ocupado.

Modelo adecuado a la herramienta N-CD++

Como se explica en el apéndice C, N-CD++ posee un concepto distinto en cuanto al manejo de los ports de los modelos DEVS y Cell-Devs.
Mientras que en la definición de los modelos, se asume que la información existente en los ports se encuentra disponible en todo momento (puede ser leído su valor en cualquier regla), en N-CD++ dicho valor debe ser requerido explícitamente, disparándose luego un evento de envío, cuya recepción debe ser tratada por un conjunto de reglas especiales.
Este tratamiento hizo que el modelo presentado anteriormente no pudiera ser implementado.
Por este motivo, se ha realizado una adecuación al mismo, que por no ser inmediata, se ha decidido especificar como un modelo alternativo.

Descripción del modelo

Cuando un auto llega al cruce, ingresa en la celda de entrada (con los valores recibidos por los ports conectados al tramo) y se le pide al controlador de congestión, que corresponde al tramo de salida indicado en el camino del auto, que envíe la información de congestión.
El auto queda esperando en la celda de entrada la recepción de la información de congestión para decidir si continúa en el mismo camino o toma uno alternativo. Para poder modelar esto se le agregó al estado del cruce un dato más llamado 'Ruteado' que indica si ya se ha recibido la información solicitada.
Una vez que la celda de entrada tomó la decisión prende la marca 'Ruteado' y permite que el auto siga su camino.
Por lo tanto, en este modelo sólo la celda por la cual el auto ingresa al cruce, realiza la función de ruteo. Las demás celdas siguen con el mismo comportamiento que el definido en modelos anteriores.

Definición

Cada celda del espacio se define como:

$$C_{0j}\ (nro\_cruce) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = < η, P^x, P^y>, donde

$\eta = 3$
P^x = { (X_1, Record), (X_2, Record), (X_3, Record) }
P^y = { (Y_1, Record), (Y_2, Record), (Y_3, Record) }

$X, Y \in N$

S: {s, phase, σqueue, σ}

donde s = (Destino, camino, nro_cruce, nro_tramo, ruteado)

Destino = { !=0 si hay un vehículo en la celda, indica el destino del mismo
0 sino.

Camino = { $\{t_1.t_2.\ldots\ldots t_n\}$ donde $ti \in N \wedge (\forall i\ (\exists\ r \in$ Tramos/ Nro_tramo(r) = ti))
0 sino.

nro_cruce $\in$ N: identificador unívoco del cruce

nro_tramo $\in$ N: identificador del tramo que se conecta a la celda de salida del cruce

Ruteado = { 1 si el vehículo ya ha sido ruteado
0 sino.

Este dato sólo se utiliza en las celdas de entrada del cruce.

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

delay = transport

d = Conversión_Demora(velocidad(maxc))
donde, maxc es la constante especificada para el cruce y representa la velocidad máxima de circulación permitida (en km/h).

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-Devs para demoras de transporte.

La definición de la función τ se realizará luego de introducir el modelo acoplado porque todas las celdas utilizan la información de los ports para calcular su nuevo estado.

El estado de una celda representa la presencia o ausencia de un vehículo, especificando el destino del mismo, el camino a seguir por el auto dentro de ella, un identificador único del cruce y el identificador del tramo al que se conecta cada celda de salida del cruce. Además se incluye dentro del estado un dato que indica si al entrar al cruce el auto ya se encuentra ruteado. Dentro del cruce sólo se permite que los vehículos avancen hacia la posición de adelante, siempre y cuando la misma esté vacía y el vehículo ya haya sido ruteado. De esta forma el vecindario necesario para establecer el nuevo estado de una celda está constituido por ella misma, la celda anterior y la siguiente.

Modelo acoplado

El modelo acoplado correspondiente al cruce (c, maxc) se define como:

Cruce(k, In, Out, maxc) = < Xlist, Ylist, I, X, Y, n, {t_1,...,t_n}, η, N, C, B, Z, select >

donde,

Ylist = { (0,i) / $0 \le i < k$}

Xlist = { (0,i) / $0 \le i < k$}

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,i), Natural> / $0 \le i < k$,
<$X_{\eta+2}$(0,i), Natural> / $i \in$ In,
<$X_{\eta+3+j}$(0,i), Natural> / $i \in$ In y $j \in$ Out }

P^y = {<$Y_{\eta+1}$(0,i), Natural> / $0 \le i < k$,
<$Y_{\eta+2}$(0,i), Natural> / $i \in$ Out,
<$Y_{\eta+3+j}$(0,i), Natural> / $i \in$ In y $j \in$ Out }

Celdas de entrada

El comportamiento es similar al modelo anterior.
La diferencia radica que en este caso, la celda le debe pedir al controlador de congestión la información.
Para ello se agregan los ports y-c-congestion$_j$.

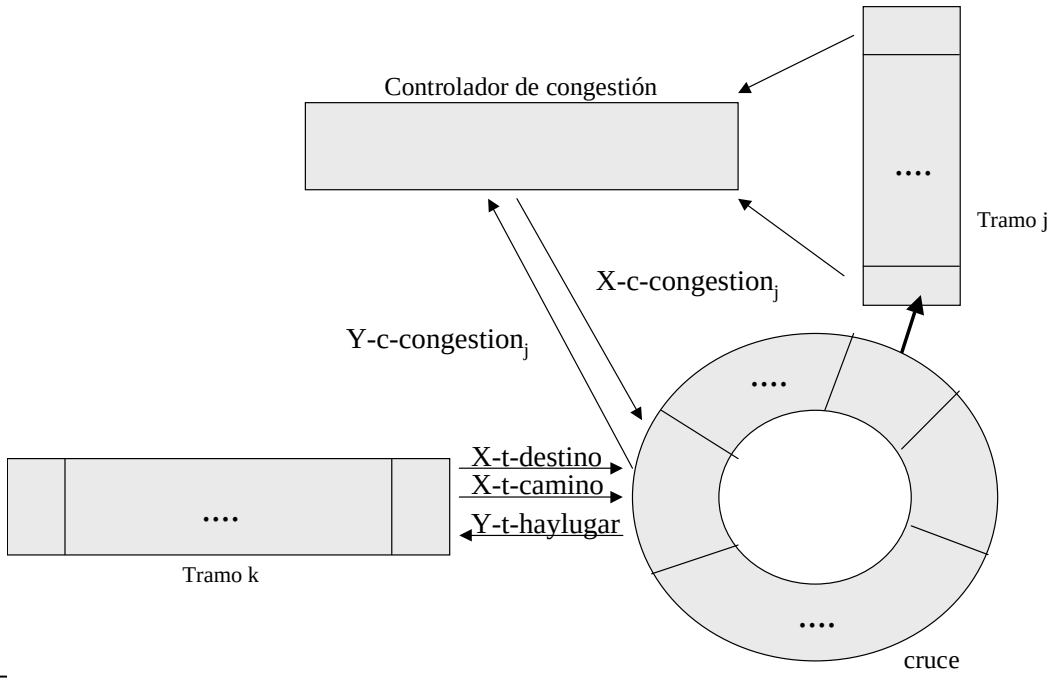


Figura 11 - Acoplamiento de una celda de entrada

PORTS		
Port	**Nombre**	**Comentario**
$X_{\eta+1}(0,i)$, $i \in$ In	x-t-destino	Este port se usa para saber si en el tramo existe un auto que desea ingresar al cruce (portvalue(x-t-destino) != 0), indicando el destino del mismo.
$X_{\eta+2}(0,i)$, $i \in$ In	x-t-Camino	Este port se usa para conocer el camino del auto que ingresa al cruce
$X_{\eta+3+j}(0,i)$, $i \in$ In Donde j es el nro de tramo al que le corresponde el crontrolador de congestión	x-c-congestion$_j$ Es decir, cada celda de entrada puede recibir información de todos los controladores de congestión que corresponden a los tramos de salida del cruce.	Por este port se conoce la información de congestión del tramo por el cual quiere salir el auto que está en el cruce.
$Y_{\eta+1}(0,i)$, $i \in$ In	y-t-haylugar	Este port informa al tramo si hay suficiente lugar en el cruce para el avance un auto.
$Y_{\eta+3+j}(0,i)$, $i \in$ In Donde j es el nro de tramo al que le corresponde el crontrolador de congestión	y-c-congestion$_j$ Es decir, cada celda de entrada le puede pedir información a todos los controladores de congestión que corresponden a los tramos de salida del cruce.	Por este port se le pide al controlador de congestión que envíe la información de congestión que posee.

Celdas de salida

No varía la interface de estas celdas.

PORTS		
Port	**Nombre**	**Comentario**
$X_{\eta+1}(0,i)$, $i \in$ Out	x-t-haylugar	Este port se usa para saber si en el tramo existe lugar para que salga un auto del cruce (portvalue(x-t-haylugar) = 0).
$Y_{\eta+1}(0,i)$, $i \in$ Out	y-t-destino	Este port informa al tramo si hay un auto que sale desde el cruce hacia él, indicando el destino.
$Y_{\eta+2}(0,i)$, $i \in$ Out	y-t-camino	Este port informa al tramo el camino del auto que sale desde el cruce hacia él.

$\boldsymbol{X} \in$ N

$\boldsymbol{Y} \in$ N

n = 1

$\mathbf{t_1} = k$

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

C = { C_{ij} / i = 0 $\wedge$ j $\in$ [0, k-1] }, donde cada C_{ij} es un componente Cell-Devs con demora.

B = {$\varnothing$}

Z se construye siguiendo la definición dada por el formalismo Cell-Devs, instanciada con el vecindario de este espacio.

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

Para definir la función τ se debe tener en cuenta que el comportamiento de las celdas difiere si se trata de celdas de ingreso o de salida del cruce, por lo tanto se define por separado para cada caso.

Celdas de ingreso al cruce

Las celdas de ingreso al cruce son:

{ (0,i) / i $\in$ In }

FUNCION τ		
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = nro_Tramo(0,0) *Ruteado* = ruteado(0,-1)	Destino(0,0) = 0 and Destino(0,-1) != 0 and ruteado(0,-1) = 1 */* Ingresa a la celda un auto que ya estaba dentro del cruce */*	Send(1, y-t-haylugar)
Destino = portvalue(x-t-destino) *Camino* = portvalue(x-t-Camino) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0) *Ruteado* = 0	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and Portvalue(x-t-destino) != nro_cruce(0,0) */* Entra al cruce un auto que todavía no ha llegado a destino */*	Send(1, y-t-haylugar) send(Pregunta, y-c-congestion $_{Proximo\_Tramo(x\text{-}t\text{-}camino)}$)
Destino = 0 *Camino*= 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0) *Ruteado* = 0	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and Portvalue(x-t-destino) = nro_cruce(0,0) */* Elimina el auto del modelo ya que ha llegado a destino */*	Send(0, y-t-haylugar)
Destino = Destino(0,0) *Camino* = Camino(0,0) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = nro_Tramo(0,0) *Ruteado* = 1	Destino(0,0) != 0 and Ruteado(0,0) = 0 and ¬Hay_Congestion(portvalue(x-c-congestion $_{Proximo\_Tramo(Camino(0,0))}$) , Proximo_Tramo(Camino(0,0))) /* En este regla se efectúa el ruteo, decidiendo que se conserva el mismo camino */	Send(1, y-t-haylugar)
Destino = Destino(0,0) *Camino* = Nuevo_Camino(Nro_Cruce(0,0), Destino(0,0), Camino(0,0))	Destino(0,0) != 0 and Ruteado(0,0) = 0 and Hay_Congestion(Portvalue(x-c-congestion $_{Proximo\_Tramo(Camino(0,0))}$) , Proximo_Tramo(Camino(0,0)))	Send(1, y-t-haylugar)

Nro_cruce = nro_cruce(0,0) *Nro_tramo* = nro_Tramo(0,0) *Ruteado* = 1	/* En este regla se efectúa el ruteo, decidiendo que se elige un camino alternativo */	
Destino = 0 *Camino* = 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0)	Destino(0,0) != 0 and Destino(0,1) = 0 and ruteado(0,0) = 1 and Destino(0,-1) = 0 /* *Avanza dentro del cruce si el auto ya ha sido ruteado. No hay un auto con prioridad dentro del cruce* */	Send(0, y-t-haylugar)
Destino = 0 *Camino* = 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0)	Destino(0,0) != 0 and Destino(0,1) = 0 and ruteado(0,0) = 1 and Destino(0,-1) != 0 /* *Avanza dentro del cruce si el auto ya ha sido ruteado. Hay un auto con prioridad dentro del cruce* */	Send(1, y-t-haylugar)
(0,0)	T /* *En otro caso la celda conserva el estado* */	-

Este tipo de celdas pueden recibir autos desde el modelo acoplado o de la celda anterior en el cruce.
Sólo recibe el vehículo de la celda anterior si éste ya ha sido ruteado (regla 1).
En el comportamiento de esta celda, se pueden identificar tres fases:

1. El auto ingresa a la celda:
   Si el auto ingresa desde la celda anterior del cruce: lo deja seguir su camino (Primer regla)
   Si el auto ingresa desde el modelo acoplado:
   Si no llegó a destino:
   Lo recibe en la celda (indicando que no se encuentra ruteado) y le pregunta al controlador de congestión la información correspondiente al tramo indicado en el camino del auto. (Segunda regla)
   Si llegó a destino:
   Lo elimina del modelo. (Tercer regla)

2. Una vez que se recibe información de congestión, decide si continuar por el camino original o salir por uno alternativo (Cuarta y quinta regla)

3. La quinta y sexta regla permiten que el vehículo avance a la próxima celda, sólo si ya se ha efectuado el ruteo del mismo.

Celdas de salida del cruce

Las celdas de salida del cruce son:

{ (0,i) / i ∈ Out }

	FUNCION τ	
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Destino = Destino(0,-1) *Camino*= Camino(0,-1) *nro_cruce*= nro_cruce(0,0) *nro_tramo*= nro_tramo(0,0) *ruteado*= ruteado(0,-1)	Destino(0,0) = 0 and Destino(0,-1) != 0 and ruteado(0,-1) = 1 and ((portvalue(x-t-haylugar) = 1) or (portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) != Nro_Tramo(0,0)))	Send(0, y-t-destino)

	/* Llega auto que permanecerá dentro del cruce */	
Destino = 0 *Camino*= 0 *nro_cruce*= nro_cruce(0,0) *nro_tramo*= nro_tramo(0,0) *ruteado*= 0	Destino(0,0) = 0 and Destino(0,-1) != 0 and ruteado(0,-1) = 1 and Portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) = Nro_Tramo(0,0) /* Llega auto que abandonará el cruce*/	Send(Destino(0,-1), y-t-destino) Send(Cola_Camino (Camino(0,-1)), y-t-camino)
Destino = 0 *Camino*= 0 *nro_cruce*= nro_cruce(0,0) *nro_tramo*= 0 *ruteado*= 0	Destino(0,0) != 0 and Destino(0,1) = 0 /* Sigue en el cruce */	Send(0, y-t-destino)
(0,0)	t /*En otro caso la celda conserva el estado */	Send(0, y-t-destino)

Las celdas de salida reciben los vehículos de la celda anterior en el cruce, pero sólo los dejan avanzar si ya se ha tomado la decisión de ruteo (ruteado(0,-1) = 1).
El comportamiento de este tipo de celdas es el mismo que para el caso de ruteo estático.

Acoplamiento entre Controlador de congestión y Tramos

Tramos de un solo carril

Un tramo se haya acoplado a un controlador de congestión a través de dos ports de salida y uno de entrada. El tramo, a través de estos ports, informa al controlador de congestión del ingreso y salida de autos del mismo. El controlador de congestión calcula la cantidad de autos que en un determinado momento se encuentran en el tramo, utilizándose esta información como medida de congestión.

Para completar la definición del acoplamiento entre controladores de congestión y tramos hay que establecer a través de que ports se comunican.

El modelo DEVS para un controlador de congestión se haya conectado a la celda (0,0) del tramo a través de un port de entrada, y a la celda (0,k-1) también a través de un port de entrada.

Se conectan los siguientes ports:

- $Y_{\eta+2}$(0,0) (y-r-EntraAuto) (Tramo) → X_1 (x-r-hayautoin) (Controlador de congestión) informando del ingreso de un auto al tramo
- $Y_{\eta+2}$(0,k-1) (y-r-SaleAuto) (Tramo) → X_2 (x-r-hayautoout) (Controlador de congestión) informando del egreso de un auto al tramo

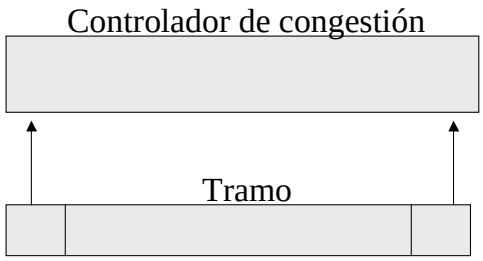


Figura 12 - Ports de acoplamiento entre tramos y controladores de congestión

Los ports se acoplan definiendo la función Z en base a la información anteriormente especificada.

Tramos de varios carriles

Primer alternativa: el controlador almacena en su estado la cantidad de autos existentes en todo el tramo.

El modelo DEVS para un controlador de congestión se haya conectado a las celdas (i,0) y (i,k-1) con $0 \leq i \leq n-1$, siendo n la cantidad de carriles del tramo, a través de sus dos ports de entrada.

Se conectan los siguientes ports:

- $Y_{\eta+2}$(i,0) (y-r-EntraAuto$_i$) (Tramo) → X_1 (x-r-hayautoin) (Controlador de congestión) informando del ingreso de un auto al tramo
- $Y_{\eta+2}$(i,k-1) (y-r-SaleAuto$_i$) (Tramo) → X_2 (x-r-hayautoout) (Controlador de congestión) informando del egreso de un auto al tramo

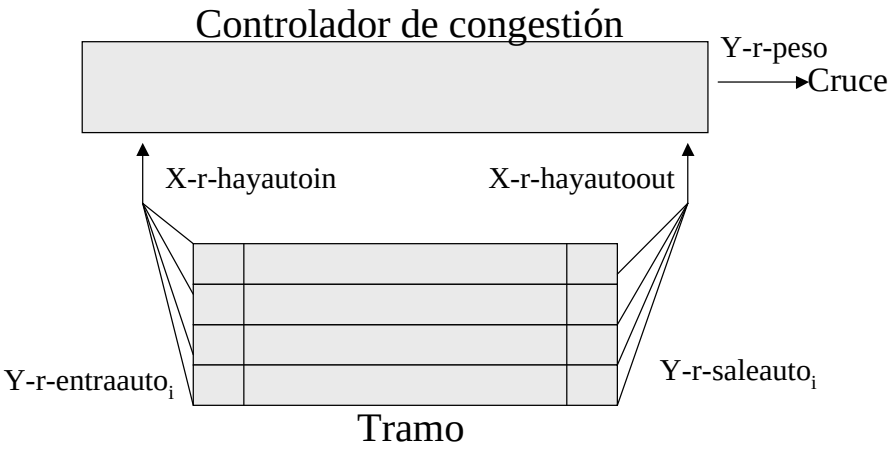


Figura 13 - Acoplamiento entre un tramo de varios carriles y el controlador de congestión

Segunda alternativa: el controlador almacena en su estado la cantidad de autos existentes en cada carril del tramo.
El modelo DEVS para un controlador de congestión se haya conectado a las celdas (i,0) y (i,k-1) con $0 \leq i \leq n-1$, siendo n la cantidad de carriles del tramo, pero en este caso posee dos ports de entrada para cada carril del tramo, monitoreando cada carril en forma independiente.
En este caso se conectan los siguientes ports:

- $Y_{\eta+2}$(i,0) (y-r-EntraAuto$_i$) (Tramo) → X_i (x-r-hayautoin$_i$) (Controlador de congestión) informando del ingreso de un auto al tramo
- $Y_{\eta+2}$(i,k-1) (y-r-SaleAuto$_i$) (Tramo) → X_{n+i} (x-r-hayautoout$_i$) (Controlador de congestión) informando del egreso de un auto al tramo

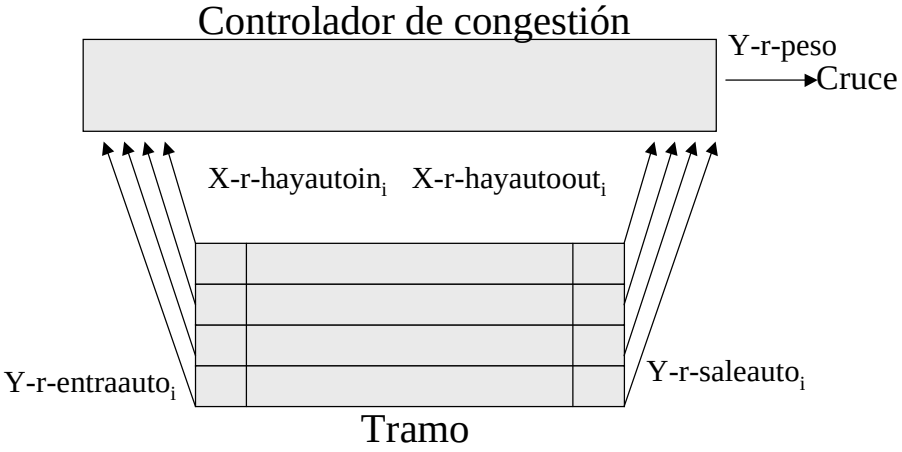


Figura 14 - Acoplamiento entre un tramo de varios carriles y el controlador de congestión

Acoplamiento entre Controlador de congestión y Cruce

De acuerdo a los modelos definidos para los cruces (tanto el original como su adecuación), el cruce se acopla a los modelos de congestión sólo a través de las celdas de entrada al mismo (ya que son las encargadas de efectuar el ruteo en base a la información de congestión).
Cada celda de entrada del cruce contendrá un port de entrada por cada tramo de salida del mismo, cada uno de estos ports se encuentra conectado al controlador de congestión que monitorea al tramo de salida.

Denominamos In al conjunto de celdas de entrada del cruce, y Out al conjunto de celdas de salida. Sea $j \in$ Out, identificaremos con T_j al tramo acoplado a la celda de salida j y con Cj al controlador de congestión de dicho tramo.

$\forall j \in$ Out: ($\forall i \in$ In: (Y_1 (y-r-peso)$_j \rightarrow$ $X_{\eta+3+j}(0,i)$ (x-c-congestion$_j$)))

Esto indica que, el port de salida del controlador de congestión Cj se conecta con todas las celdas de entrada del cruce, a través del port de entrada identificado como η+3+j

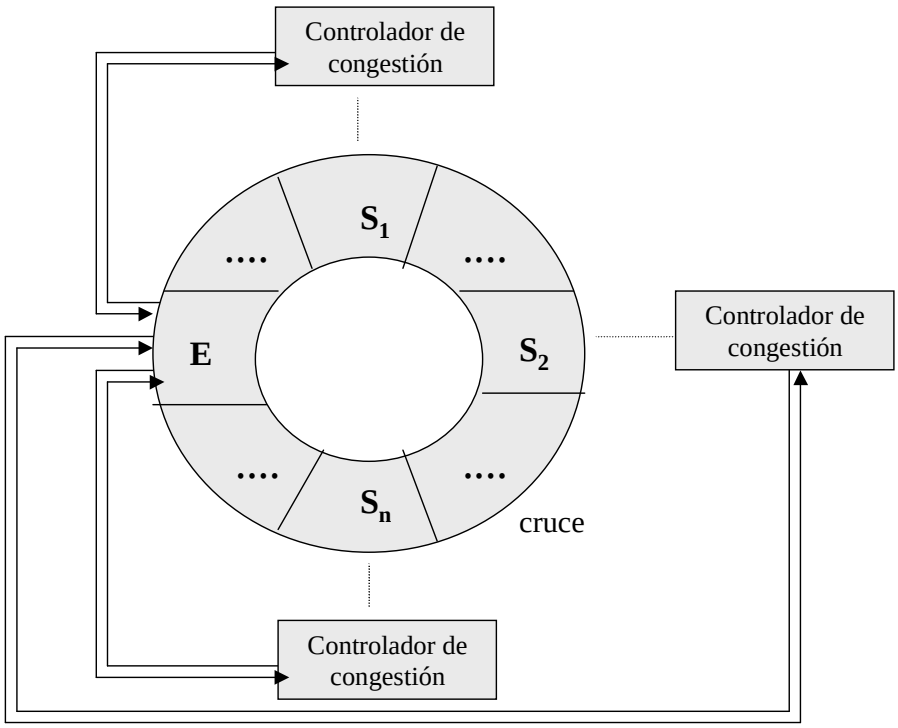


Figura 15 - Acoplamiento entre el cruce y los controladores de congestión

Acoplamiento entre Tramos y Cruces

El acoplamiento entre cruces y tramos no varía con respecto al modelo presentado para el caso de ruteo estático.

Sección 4.3: Implementación de algunos ejemplos

Ejemplo de Ruteo estático

En esta sección exponemos un ejemplo sencillo en el cuál se muestra el funcionamiento del ruteo estático. El ejemplo fue implementado utilizando la herramienta N-CD++ [RW99].
En la siguiente figura podemos observar un esquema del ejemplo que se va a describir.

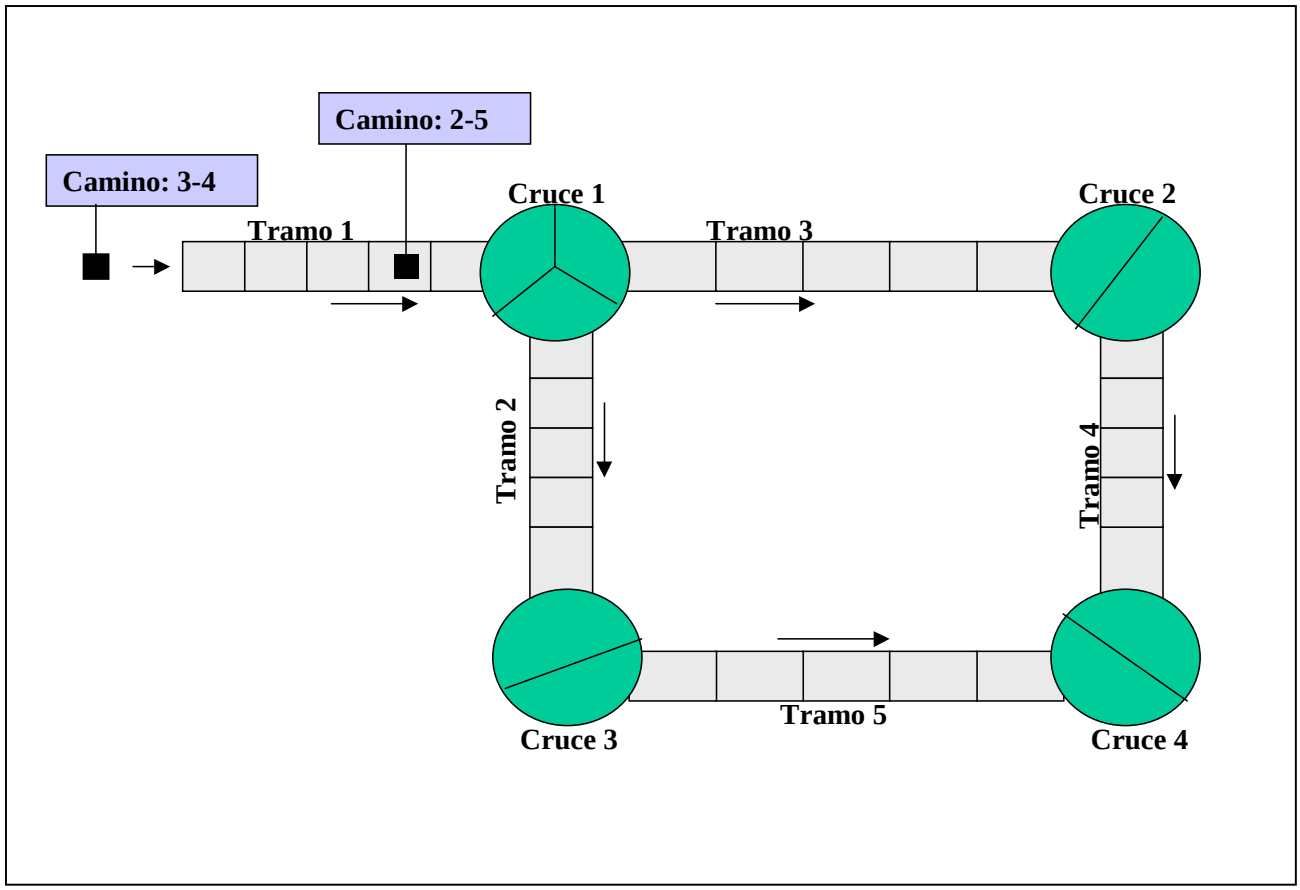


Figura 1 – Esquema del modelo

El modelo ejemplificado está formado por 5 tramos y cuatro cruces conectados de la forma en que se observa en la figura 1. Se ubicaron en el mismo dos autos, que seguirán distintos caminos. El objetivo del presente ejemplo es únicamente el de explicar el funcionamiento del ruteo estático, de ahí la simplicidad del mismo.

La implementación de los modelos de tramos y cruces varía con respecto a la presentada en la especificación debido en general a ciertas restricciones, que por el momento, presenta la herramienta N-CD++ .
A continuación se detallan las diferencias existentes con la especificación presentada en la sección de ruteo estático.

- Los tramos fueron implementados como modelos Cell_Devs de dos dimensiones, y no de una dimensión como figura en la especificación. La segunda dimensión se utiliza para representar el estado de la celda. Cada nivel de la segunda dimensión representa una parte del estado de la celda. El nivel 0 de la segunda dimensión representa si hay o no auto en la celda. El nivel 1 representa el camino que seguirá el auto. La razón de este cambio en la implementación es que en la herramienta N-CD++ el estado de las celdas únicamente puede representarse con un número real y en nuestro caso el estado de los tramos es una estructura.

- Los cruces fueron implementados como modelos Cell_Devs de dos dimensiones, y no de una dimensión como figura en la especificación. La segunda dimensión se utiliza para representar el estado de la celda. Cada nivel de la segunda dimensión representa una parte del estado de la celda. El nivel 0 de la dimensión 2 representa si hay o no auto en la celda. El nivel 1 de la dimensión 2 representa el camino que seguirá el auto. El nivel 2 contiene la identificación del cruce. El nivel 3 contiene la identificación de los tramos a los cuales se hallan conectadas las celdas de salida del cruce. La razón de este cambio en la implementación es que en la herramienta N-CD++ el estado de las celdas únicamente puede representarse con un número real y en nuestro caso el estado de los tramos es una estructura.

- Para simplificar la implementación, el camino fue representado por un número real con el siguiente formato:

  d . d d d d d

  donde
  1=< d <=9 representa el identificador de un tramo. El dígito de la parte entera representa el primer tramo del camino y la parte fraccionaria el resto del camino.
  Según la especificación el camino se define como una lista de tramos, de modo que una implementación más correcta del mismo sería una lista o un arreglo (de tramos), pero la herramienta N-CD++ aún no soporta este tipo de representaciones en el estado de las celdas.

La implementación completa del modelo puede consultarse en el Apéndice C.

Los resultados luego de ejecutar el modelo hasta la finalización de la simulación fueron los siguientes:
Los autos recorren en primer lugar el tramo 1.

Resultado de la simulación – Tramo1

```
Time: 00:00:00:000
     0    1    2    3    4
  +-------------------------+
0 |                   1.00  |
1 |                   2.50  |
  +-------------------------+

Time: 00:00:00:010
     0    1    2    3    4
  +-------------------------+
0 | 1.00                1.00|
1 | 3.40                2.50|
  +-------------------------+

Time: 00:00:00:020
     0    1    2    3    4
  +-------------------------+
0 |      1.00               |
1 |      3.40               |
  +-------------------------+

Time: 00:00:00:030
     0    1    2    3    4
  +-------------------------+
0 |           1.00          |
1 |           3.40          |
  +-------------------------+

Time: 00:00:00:040
     0    1    2    3    4
  +-------------------------+
0 |                   1.00  |
1 |                   3.40  |
  +-------------------------+

Time: 00:00:00:050
     0    1    2    3    4
  +-------------------------+
0 |                   1.00  |
1 |                   3.40  |
  +-------------------------+

Time: 00:00:00:060
     0    1    2    3    4
  +-------------------------+
0 |                         |
1 |                         |
  +-------------------------+
```

El nivel 0 de la dimensión 2 del tramo indica si hay un auto o no en la celda. El nivel 1 de la dimensión 2 del tramo representa el camino que seguirá el auto que se encuentra en la celda. Por ejemplo, en el tiempo 00:00:00:010, se observa que hay dos autos en el tramo 1 en las celdas (0,0) y (0,4). El auto que se encuentra en la celda (0,0) seguirá el camino *'Tramo3 - Tramo4'* (es decir aquel que pase por los tramos 3 y 4), mientras que el auto que se encuentra en la celda (0,4) seguirá el camino *'Tramo2 – Tramo5'*.

Como se observa en la figura del ejemplo 1, al finalizar el recorrido del tramo 1 los autos atraviesan el cruce 1.

Resultado de la simulación -Cruce1

```
Time: 00:00:00:000
     0        1        2
  +-------------------------+
0 |                         |
1 |                         |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+

Time: 00:00:00:010
     0        1        2
  +-------------------------+
0 |                         |
1 |                         |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+

Time: 00:00:00:020
     0        1        2
  +-------------------------+
0 |  1.00                   |
1 |  2.50                   |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+

Time: 00:00:00:030
     0        1        2
  +-------------------------+
0 |                         |
1 |                         |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+

Time: 00:00:00:050
     0        1        2
  +-------------------------+
0 |                         |
1 |                         |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+

Time: 00:00:00:060
     0        1        2
  +-------------------------+
0 |  1.00                   |
1 |  3.40                   |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+

Time: 00:00:00:070
     0        1        2
  +-------------------------+
0 |          1.00          |
1 |          3.40          |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+

Time: 00:00:00:080
     0        1        2
  +-------------------------+
0 |                         |
1 |                         |
2 |  1.00    1.00    1.00   |
3 |          2.00    3.00   |
  +-------------------------+
```

El nivel 0 de la dimensión 2 del cruce indica si hay un auto o no en la celda. El nivel 1 de la dimensión 2 representa el camino que seguirá el auto que se encuentra en la celda. El nivel 2 de la dimensión 2 del cruce contiene una identificación unívoca del mismo y el nivel 3 indica el tramo al que se halla conectada la celda. Por ejemplo, vemos que el cruce 1 contiene un 2 en la celda (3,1) ya que la misma se halla conectada al tramo 2. Como fue explicado en la especificación del ruteo estático, el cruce es el encargado de realizar el ruteo, es decir de derivar los autos a los tramos correspondientes, según el camino que los mismos deben tomar. El cruce determina el tramo por el cuál debe salir el auto, pero no incluye el mismo en el camino asociado al auto. En otras palabras se elimina del camino el tramo por el cual el cruce decide enviar el auto. Esto es válido ya que para que el ruteo de un auto se realice correctamente, los cruces únicamente necesitan conocer el trayecto que le falta recorrer al mismo, pudiendo prescindir de la información de los tramos ya recorridos.

El cruce 1 rutea el auto cuyo camino es ***'Tramo 2–Tramo 5'*** al tramo 2 y el auto cuyo camino es ***'Tramo 3 – Tramo 4'*** al tramo 3. Esto puede observarse en los resultados de la simulación para los tramos 2 y 3:

Resultado de la simulación -Tramo2

```
Time: 00:00:00:000
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+

Time: 00:00:00:020
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+

Time: 00:00:00:030
    0   1   2   3   4
  +-------------------+
0 | 1.00              |
1 | 5.00              |
  +-------------------+

Time: 00:00:00:040
    0   1   2   3   4
  +-------------------+
0 |     1.00          |
1 |     5.00          |
  +-------------------+

Time: 00:00:00:050
    0   1   2   3   4
  +-------------------+
0 |     1.00          |
1 |     5.00          |
  +-------------------+

Time: 00:00:00:060
    0   1   2   3   4
  +-------------------+
0 |         1.00      |
1 |         5.00      |
  +-------------------+

Time: 00:00:00:070
    0   1   2   3   4
  +-------------------+
0 |             1.00  |
1 |             5.00  |
  +-------------------+

Time: 00:00:00:080
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+
```

En el tiempo 00:00:00:030 de la salida del tramo 2, observamos la entrada del auto cuyo camino era inicialmente ***'Tramo 2 – Tramo 5'***. Ahora el camino fue truncado y solo se guarda, en el estado, el trayecto que le falta recorrer (tramo5) sin contar el tramo actual.

Resultado de la simulación –Tramo3

```
Time: 00:00:00:000
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+

Time: 00:00:00:070
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+

Time: 00:00:00:080
    0   1   2   3   4
  +-------------------+
0 | 1.00              |
1 | 4.00              |
  +-------------------+

Time: 00:00:00:090
    0   1   2   3   4
  +-------------------+
0 |     1.00          |
1 |     4.00          |
  +-------------------+

Time: 00:00:00:100
    0   1   2   3   4
  +-------------------+
0 |     1.00          |
1 |     4.00          |
  +-------------------+

Time: 00:00:00:110
    0   1   2   3   4
  +-------------------+
0 |         1.00      |
1 |         4.00      |
  +-------------------+

Time: 00:00:00:120
    0   1   2   3   4
  +-------------------+
0 |             1.00  |
1 |             4.00  |
  +-------------------+

Time: 00:00:00:130
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+
```

De igual forma, en el tiempo 00:00:00:080 del resultado de la simulación para el tramo 3, observamos la entrada del auto cuyo camino era inicialmente ***'Tramo 3 – Tramo 4'***. Ahora el camino fue truncado y solo se guarda, en el estado, el trayecto que le falta recorrer (tramo 4) sin contar el tramo actual.

Como podemos observar en la figura, el auto cuyo camino inicial era ***'Tramo2 - Tramo5'*** atraviesa el cruce 3 que se encarga de rutear el mismo hacia el tramo 5.

Resultado de la simulación – Cruce3

```
Time: 00:00:00:000
      0     1
  +---------------+
0 |               |
1 |               |
2 |  3.00  3.00   |
3 |        5.00   |
  +---------------+

Time: 00:00:00:070
      0     1
  +---------------+
0 |               |
1 |               |
2 |  3.00  3.00   |
3 |        5.00   |
  +---------------+

Time: 00:00:00:080
      0     1
  +---------------+
0 |  1.00         |
1 |  5.00         |
2 |  3.00  3.00   |
3 |        5.00   |
  +---------------+
```

```
+------------------+        +------------------+        +------------------+
```

El cruce 3 posee una celda de entrada (celda 0) conectada al tramo 2 y una celda de salida (celda 1) conectada al tramo 5. Por la celda 0 ingresa en el tiempo 00:00:00:080, el auto cuyo camino inicial era '***Tramo2-Tramo5',*** y el mismo sale por la celda 1 hacia el tramo 5.

Resultado de la simulación – Tramo5

```
Time: 00:00:00:000
    0   1   2   3   4
  +-------------------------+
0 |                         |
1 |                         |
  +-------------------------+

Time: 00:00:00:080
    0   1   2   3   4
  +-------------------------+
0 |                         |
1 |                         |
  +-------------------------+

Time: 00:00:00:090
    0   1   2   3   4
  +-------------------------+
0 | 1.00                    |
1 | -1.00                   |
  +-------------------------+

Time: 00:00:00:100
    0   1   2   3   4
  +-------------------------+
0 |     1.00                |
1 |    -1.00                |
  +-------------------------+

Time: 00:00:00:110
    0   1   2   3   4
  +-------------------------+
0 |         1.00            |
1 |        -1.00            |
  +-------------------------+

Time: 00:00:00:120
    0   1   2   3   4
  +-------------------------+
0 |             1.00        |
1 |            -1.00        |
  +-------------------------+

Time: 00:00:00:130
    0   1   2   3   4
  +-------------------------+
0 |                 1.00    |
1 |                -1.00    |
  +-------------------------+

Time: 00:00:00:140
    0   1   2   3   4
  +-------------------------+
0 |                         |
1 |                         |
  +-------------------------+
```

Observamos en el resultado de la simulación del tramo 5, la entrada del auto cuyo camino inicial era '***Tramo2 - Tramo 5*** (en el tiempo 00:00:00:090).
En el nivel 1 de la dimensión 2 se observa el valor –1 que indica que el tramo actual es el último que el auto recorrerá, es decir que el auto llegará a destino al ingresar al próximo cruce.

Continuando con el análisis de la salida de la simulación, el auto cuyo camino inicial era ***'Tramo3-Tramo 4'*** atraviesa el cruce 2 que se encarga de rutear el mismo hacia el tramo 4.

Resultado de la simulación – Cruce2

```
Time: 00:00:00:000
      0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  2.00    2.00     |
3 |          4.00     |
  +-------------------+

Time: 00:00:00:120
      0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  2.00    2.00     |
3 |          4.00     |
  +-------------------+

Time: 00:00:00:130
      0       1
  +-------------------+
0 |  1.00             |
1 |  4.00             |
2 |  2.00    2.00     |
3 |          4.00     |
  +-------------------+

Time: 00:00:00:140
      0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  2.00    2.00     |
3 |          4.00     |
  +-------------------+
```

El cruce 2 posee una celda de entrada (celda 0) conectada al tramo 3 y una celda de salida (celda 1) conectada al tramo 4. Por la celda 0 ingresa en el tiempo 00:00:00:130, el auto cuyo camino inicial era '***Tramo3-Tramo4',*** y el mismo sale por la celda 1 hacia el tramo 4.

Resultado de la simulación – Tramo4

```
Time: 00:00:00:000
     0   1   2   3   4
  +-------------------------+
0 |                         |
1 |                         |
  +-------------------------+

Time: 00:00:00:130
     0   1   2   3   4
  +-------------------------+
0 |                         |
1 |                         |
  +-------------------------+

Time: 00:00:00:140
     0   1   2   3   4
  +-------------------------+
0 | 1.00                    |
1 | -1.00                   |
  +-------------------------+

Time: 00:00:00:150
     0   1   2   3   4
  +-------------------------+
0 |      1.00               |
1 |     -1.00               |
  +-------------------------+

Time: 00:00:00:160
     0   1   2   3   4
  +-------------------------+
0 |          1.00           |
1 |         -1.00           |
  +-------------------------+

Time: 00:00:00:170
     0   1   2   3   4
  +-------------------------+
0 |              1.00       |
1 |             -1.00       |
  +-------------------------+

Time: 00:00:00:180
     0   1   2   3   4
  +-------------------------+
0 |                   1.00  |
1 |                  -1.00  |
  +-------------------------+

Time: 00:00:00:190
     0   1   2   3   4
  +-------------------------+
0 |                         |
1 |                         |
  +-------------------------+
```

Observamos en el resultado de la simulación del tramo 4, la entrada del auto cuyo camino inicial era ***'Tramo 3 – Tramo 4'***.
En el nivel 1 de la dimensión 2 se observa el valor –1 que indica que el tramo actual es el último que el auto recorrerá, es decir que el auto llegará a destino al ingresar al próximo cruce.

Finalmente, mostramos el resultado del cruce 4. En los tiempos 00:00:00:140 y 00:00:00:190 se observan las llegadas de los dos autos.

Resultado de la simulación – Cruce4

```
Time: 00:00:00:000
        0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+

Time: 00:00:00:130
        0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+

Time: 00:00:00:140
        0       1
  +-------------------+
0 |  1.00             |
1 | -1.00             |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+

Time: 00:00:00:150
        0       1
  +-------------------+
0 |             1.00  |
1 |            -1.00  |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+

Time: 00:00:00:160
        0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+

Time: 00:00:00:190
        0       1
  +-------------------+
0 |  1.00             |
1 | -1.00             |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+

Time: 00:00:00:200
        0       1
  +-------------------+
0 |             1.00  |
1 |            -1.00  |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+

Time: 00:00:00:210
        0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00    4.00     |
3 |                   |
  +-------------------+
```

Ejemplo de Ruteo dinámico

En esta sección exponemos un ejemplo sencillo en el cuál se muestra el funcionamiento del ruteo dinámico. El ejemplo fue implementado utilizando la herramienta N-CD++.
En la siguiente figura podemos observar un esquema del ejemplo que se va a describir:

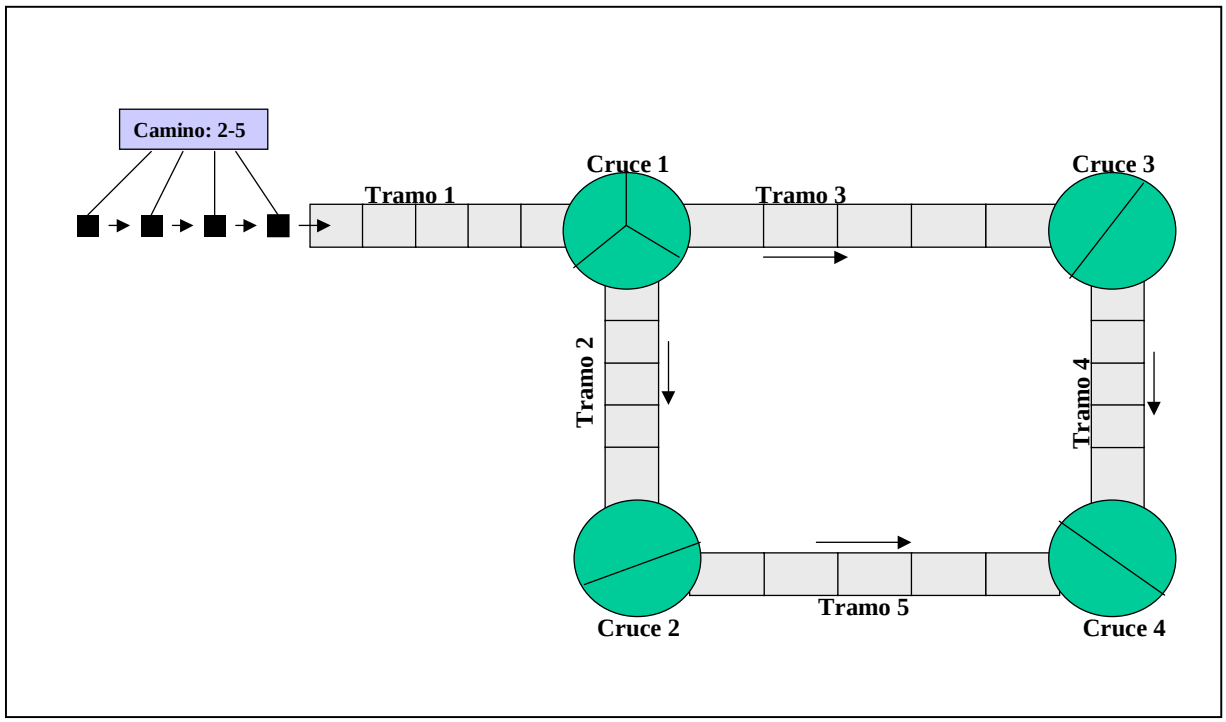


Figura 2 – Esquema del modelo

El modelo ejemplificado está formado por cinco tramos y cuatro cruces conectados de la forma en que se observa en la figura 1. Ingresan al modelos cuatro autos que desean seguir el mismo camino (*Tramo2-Tramo5*). El tramo 2 posee un bache en la celda 3 del mismo, por lo tanto, hace que los autos se demoren al pasar por el mismo. Para mantener el modelo lo más simple posible, se ha considerado que hay congestión si existen 2 o más autos en el tramo (para ejemplos reales la función que indica congestión debería ser más exigente).
El objetivo del presente ejemplo es únicamente el de explicar el funcionamiento del ruteo dinámico, de ahí la simplicidad del mismo.
La implementación de los modelos de tramos y cruces varía con respecto a la presentada en la especificación debido en general a ciertas restricciones, que por el momento, presenta la herramienta N-CD++. El modelo base que se ha tomado es el que posee en el estado del cruce una marca de ruteado (ver sección de ruteo dinámico).
A continuación se detallan las diferencias existentes con la especificación presentada en la sección de ruteo dinámico.

- Los tramos fueron implementados como modelos Cell_Devs de dos dimensiones, y no de una dimensión como figura en la especificación. La segunda dimensión se utiliza para representar el estado de la celda. Cada nivel de la segunda dimensión

representa una parte del estado de la celda. El nivel 0 de la segunda dimensión contiene el destino del auto en la celda (contiene 0 si está libre). El nivel 1 representa el camino que seguirá el auto. La razón de este cambio en la implementación es que en la herramienta N-CD++ el estado de las celdas únicamente puede representarse con un número real y en nuestro caso el estado de los tramos es una estructura.

- Los cruces fueron implementados como modelos Cell_Devs de dos dimensiones, y no de una dimensión como figura en la especificación. La segunda dimensión se utiliza para representar el estado de la celda. Cada nivel de la segunda dimensión representa una parte del estado de la celda. El nivel 0 de la dimensión 2 contiene el destino del auto en la celda. El nivel 1 de la dimensión 2 representa el camino que seguirá el auto. El nivel 2 contiene la identificación del cruce. El nivel 3 contiene la identificación de los tramos a los cuales se hallan conectadas las celdas de salida del cruce. El nivel 5 se usa sólo para las celdas de entrada y contiene la marca de ruteado. La razón de este cambio en la implementación es que en la herramienta N-CD++ el estado de las celdas únicamente puede representarse con un número real y en nuestro caso el estado de los tramos es una estructura.

- Para simplificar la implementación, el camino fue representado por un número real con el siguiente formato:

  d . d d d d d

  donde
  1=< d <=9 representa el identificador de un tramo. El dígito de la parte entera representa el primer tramo del camino y la parte fraccionaria el resto del camino.
  Según la especificación el camino se define como una lista de tramos, de modo que una implementación más correcta del mismo sería una lista o un arreglo (de tramos), pero la herramienta N-CD++ aún no soporta este tipo de representaciones en el estado de las celdas.
  Si se tienen más de nueve tramos se puede representar el camino como
  d^n . d^n d^n d^n d^n, donde n es la cantidad de dígitos que se necesitan para la identificación de los tramos.

- Se adaptaron las reglas que leen los ports de entrada, al concepto usado en la herramienta N-CD++. Para ver la explicación detallada de este cambio, ver el apéndice C.

La implementación completa del modelo puede consultarse en el Apéndice B.

Los resultados luego de ejecutar el modelo hasta la finalización de la simulación fueron los siguientes:

Los autos recorren en primer lugar el tramo 1.

Resultado de la simulación – Tramo1

```
Time: 00:00:00:000
     0    1    2    3    4
 +-------------------------+
0|                         |
1|                         |
 +-------------------------+

Time: 00:00:00:010
     0    1    2    3    4
 +-------------------------+
0|4.00                     |
1|2.50                     |
 +-------------------------+

Time: 00:00:00:020
     0    1    2    3    4
 +-------------------------+
0|     4.00                |
1|     2.50                |
 +-------------------------+

Time: 00:00:00:030
     0    1    2    3    4
 +-------------------------+
0|          4.00           |
1|          2.50           |
 +-------------------------+

Time: 00:00:00:040
     0    1    2    3    4
 +-------------------------+
0|4.00           4.00      |
1|2.50           2.50      |
 +-------------------------+

Time: 00:00:00:050
     0    1    2    3    4
 +-------------------------+
0|     4.00           4.00|
1|     2.50           2.50|
 +-------------------------+

Time: 00:00:00:060
     0    1    2    3    4
 +-------------------------+
0|          4.00           |
1|          2.50           |
 +-------------------------+

Time: 00:00:00:070
     0    1    2    3    4
 +-------------------------+
0|4.00           4.00      |
1|2.50           2.50      |
 +-------------------------+

Time: 00:00:00:080
     0    1    2    3    4
 +-------------------------+
0|     4.00           4.00|
1|     2.50           2.50|
 +-------------------------+

Time: 00:00:00:090
     0    1    2    3    4
 +-------------------------+
0|          4.00           |
1|          2.50           |
 +-------------------------+

Time: 00:00:00:100
     0    1    2    3    4
 +-------------------------+
0|4.00           4.00      |
1|2.50           2.50      |
 +-------------------------+

Time: 00:00:00:110
     0    1    2    3    4
 +-------------------------+
0|     4.00           4.00|
1|     2.50           2.50|
 +-------------------------+

Time: 00:00:00:120
     0    1    2    3    4
 +-------------------------+
0|          4.00           |
1|          2.50           |
 +-------------------------+

Time: 00:00:00:130
     0    1    2    3    4
 +-------------------------+
0|               4.00      |
1|               2.50      |
 +-------------------------+

Time: 00:00:00:140
     0    1    2    3    4
 +-------------------------+
0|                    4.00|
1|                    2.50|
 +-------------------------+

Time: 00:00:00:150
     0    1    2    3    4
 +-------------------------+
0|                         |
1|                         |
 +-------------------------+
```

En los tiempos 00:00:00:010, 00:00:00:040, 00:00:00:070 y 00:00:00:100, se observa que ingresan autos en el tramo 1 en la celdas (0,0). Todos los autos ingresados tienen como camino *'Tramo2 - Tramo5'* (es decir aquel que pase por los tramos 2 y 5).

Como se observa en la figura del ejemplo, al finalizar el recorrido del tramo 1 los autos atraviesan al cruce 1.

Resultado de la simulación -Cruce1

```
Time: 00:00:00:000
     0      1      2
 +---------------------+
0|                     |
1|                     |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                     |
 +---------------------+

Time: 00:00:00:050
     0      1      2
 +---------------------+
0|4.0000               |
1|2.5000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                     |
 +---------------------+

Time: 00:00:00:060
     0      1      2
 +---------------------+
0|4.0000               |
1|2.5000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|1.0000               |
 +---------------------+

Time: 00:00:00:070
     0      1      2
 +---------------------+
0|                     |
1|                     |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                     |
 +---------------------+

Time: 00:00:00:080
     0      1      2
 +---------------------+
0|4.0000               |
1|2.5000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                     |
 +---------------------+

Time: 00:00:00:090
     0      1      2
 +---------------------+
0|4.0000               |
1|2.5000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|1.0000               |
 +---------------------+

Time: 00:00:00:100
     0      1      2
 +---------------------+
0|                     |
1|                     |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                     |
 +---------------------+

Time: 00:00:00:110
     0      1      2
 +---------------------+
0|4.0000               |
1|2.5000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                     |
 +---------------------+

Time: 00:00:00:120
     0      1      2
 +---------------------+
0|4.0000               |
1|3.4000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|1.0000               |
 +---------------------+

Time: 00:00:00:130
     0      1      2
 +---------------------+
0|       4.0000       |
1|       3.4000       |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|       1.0000       |
 +---------------------+

Time: 00:00:00:140
     0      1      2
 +---------------------+
0|4.0000               |
1|2.5000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                     |
 +---------------------+

Time: 00:00:00:150
     0      1      2
 +---------------------+
0|4.0000               |
1|3.4000               |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|1.0000               |
```

```
Time: 00:00:00:160
      0      1      2
 +-----------------------+
0|       4.0000        |
1|       3.4000        |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|       1.0000        |
 +-----------------------+

Time: 00:00:00:170
      0      1      2
 +-----------------------+
0|                       |
1|                       |
2|1.0000 1.0000 1.0000 |
3|       2.0000 3.0000 |
4|                       |
 +-----------------------+
```

En este cruce se ejemplifica el ruteo dinámico.
La celda 0 del cruce es una celda de entrada, mientras que las dos restantes son de salida.
Los dos primeros autos que ingresan al cruce (tiempos 00:00:00:050 y 00:00:00:080), conservan el mismo camino con el que venían (*Tramo2-Tramo5*). Por lo tanto, como la celda 1 del cruce se encuentra conectada al tramo 2, el auto ingresa en el mismo.
Los dos últimos autos que ingresan al cruce (tiempos 00:00:00:110 y 00:00:00:140), deciden tomar un nuevo camino, ya que de acuerdo a la función de congestión el tramo 2 se encuentra congestionado. Por lo tanto, le piden a la matriz OD un camino alternativo, la cual devuelve el camino (*Tramo3-Tramo4*). Luego ingresan a la celda 1 del cruce (ya que por ella no corresponde salir) y por último salen a través de la celda 2 que se encuentra conectada al tramo 3.
Notar que:

- Los autos permanecen en la celda de entrada mientras la marca de ruteado (nivel 4) sea igual a 0, una vez que cambia a 1 el auto realiza puede realizar un movimiento.
- Sólo la celda de entrada usa el nivel 4 (marca de ruteado)
- Sólo las celdas de salida usan el nivel 3 (identificación del tramo al cual están conectadas)

Resultado de la simulación – Tramo2

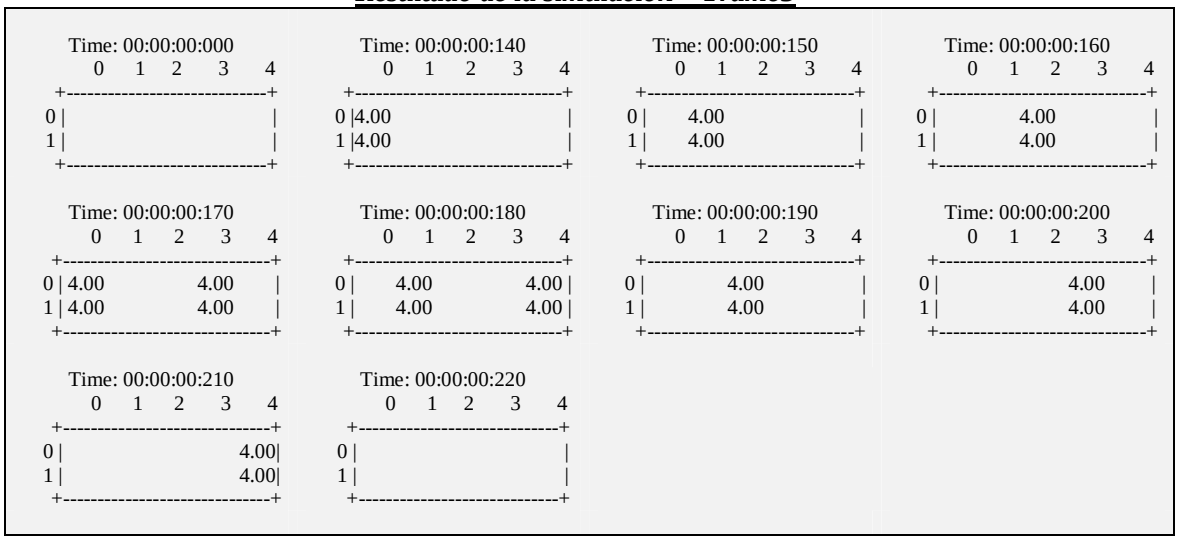
```
Time: 00:00:00:000
      0  1  2  3  4
 +-----------------+
0|                 |
1|                 |
 +-----------------+

Time: 00:00:00:070
      0  1  2  3  4
 +-----------------+
0|4.00             |
1|5.00             |
 +-----------------+

Time: 00:00:00:080
      0  1  2  3  4
 +-----------------+
0|  4.00           |
1|  5.00           |
 +-----------------+

Time: 00:00:00:090
      0  1  2  3  4
 +-----------------+
0|     4.00        |
1|     5.00        |
 +-----------------+

Time: 00:00:00:100
      0  1  2  3  4
 +-----------------+
0|4.00        4.00 |
1|5.00        5.00 |
 +-----------------+

Time: 00:00:00:110
      0  1  2  3  4
 +-----------------+
0|  4.00      4.00 |
1|  5.00      5.00 |
 +-----------------+

Time: 00:00:00:120
      0  1  2  3  4
 +-----------------+
0|     4.00 4.00   |
1|     5.00 5.00   |
 +-----------------+

Time: 00:00:00:150
      0  1  2  3  4
 +-----------------+
0|     4.00   4.00 |
1|     5.00   5.00 |
 +-----------------+

Time: 00:00:00:160
      0  1  2  3  4
 +-----------------+
0|        4.00     |
1|        5.00     |
 +-----------------+

Time: 00:00:00:210
      0  1  2  3  4
 +-----------------+
0|              4.00|
1|              5.00|
 +-----------------+

Time: 00:00:00:220
      0  1  2  3  4
 +-----------------+
0|                 |
1|                 |
 +-----------------+
```

Los dos primeros autos ingresados a la simulación siguen el camino (*Tramo 2 - Tramo5*), decisión tomada en el cruce1. Por lo tanto, ingresan al tramo en los tiempos 00:00:00:070 y 00:00:00:100 (luego de dejar el cruce 1).
Notar que:

- El camino se va consumiendo, una vez que se ingresó a un tramo ya no es necesario conservar su identificación dentro del camino. El valor 5.0 en el nivel que representa el camino, indica que el próximo tramo es el nro 5 y que allí finaliza el camino.
- En la celda 3 del tramo hay un bache, por lo tanto, los vehículos se demoran un tiempo mayor al normal en la misma.

Como se observa en la figura del ejemplo, al finalizar el recorrido del tramo 2 los autos atraviesan al cruce 2.

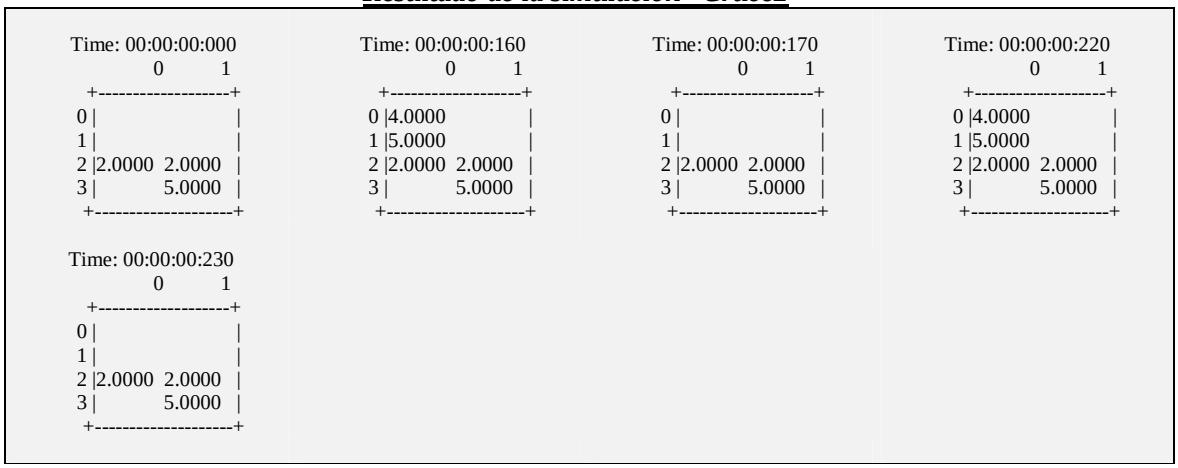
Resultado de la simulación – Tramo3



Los dos últimos autos ingresados a la simulación siguen el camino (*Tramo 3 - Tramo 4*), decisión tomada en el cruce1. Por lo tanto, ingresan al tramo en los tiempos 00:00:00:140 y 00:00:00:170 (luego de dejar el cruce 1).

Como se observa en la figura del ejemplo, al finalizar el recorrido del tramo 3 los autos atraviesan al cruce 3.

Resultado de la simulación –Cruce2



Este cruce sólo posee una celda de entrada (celda 0) y una de salida (celda 1). Por lo tanto, las únicas acciones posibles son la entrada y salida de vehículos (no toma ninguna decisión de ruteo).
Recibe los dos primeros autos ingresados a la simulación, por el tramo 2 en los tiempos 00:00:00:160 y 00:00:00:220 y los envía hacia el tramo 5 en el primer intento de entrada al mismo, ya que la primer celda del tramo se encuentra libre.
Notar que la parte entera del camino (posición (1,0) del gráfico) es igual a la identificación del tramo (posición (3,1) del gráfico), esta condición debe darse para que el cruce envíe el auto por el tramo.

Resultado de la simulación – Tramo5

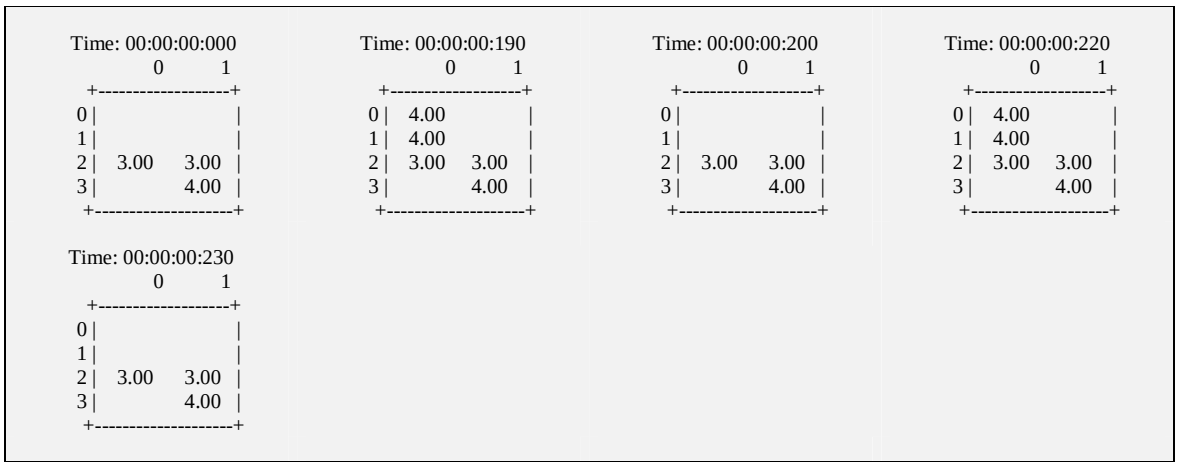
```
Time: 00:00:00:000
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+

Time: 00:00:00:160
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+

Time: 00:00:00:170
    0   1   2   3   4
  +-------------------+
0 |4.00               |
1 |-1.00               |
  +-------------------+

Time: 00:00:00:180
    0   1   2   3   4
  +-------------------+
0 |    4.00           |
1 |   -1.00           |
  +-------------------+

Time: 00:00:00:190
    0   1   2   3   4
  +-------------------+
0 |        4.00       |
1 |       -1.00       |
  +-------------------+

Time: 00:00:00:200
    0   1   2   3   4
  +-------------------+
0 |            4.00  |
1 |           -1.00  |
  +-------------------+

Time: 00:00:00:210
    0   1   2   3   4
  +-------------------+
0 |                4.00|
1 |               -1.00|
  +-------------------+

Time: 00:00:00:220
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+

Time: 00:00:00:230
    0   1   2   3   4
  +-------------------+
0 |4.00               |
1 |-1.00               |
  +-------------------+

Time: 00:00:00:240
    0   1   2   3   4
  +-------------------+
0 |    4.00           |
1 |   -1.00           |
  +-------------------+

Time: 00:00:00:250
    0   1   2   3   4
  +-------------------+
0 |        4.00       |
1 |       -1.00       |
  +-------------------+

Time: 00:00:00:260
    0   1   2   3   4
  +-------------------+
0 |            4.00  |
1 |           -1.00  |
  +-------------------+

Time: 00:00:00:270
    0   1   2   3   4
  +-------------------+
0 |                4.00|
1 |               -1.00|
  +-------------------+

Time: 00:00:00:280
    0   1   2   3   4
  +-------------------+
0 |                   |
1 |                   |
  +-------------------+
```

Los dos primeros autos ingresados a la simulación seguían el camino (*Tramo 2 - Tramo 5*), decisión tomada en el cruce1 (y luego no hay un cruce con capacidad de ruteo). Por lo tanto, ingresan a este tramo en los tiempos 00:00:00:170 y 00:00:00:230 (luego de dejar el cruce 1).
Notar que:

- Los autos se distanciaron como consecuencia del bache en el tramo 2.
- Como el camino fue consumido totalmente, se setea un valor especial (-1), a través del cual se especifica esta situación. Esto se hizo, para que las reglas correspondientes al nivel del camino detecten la finalización del mismo. El seteo de este valor no sería necesario si la herramienta aceptara en su estado un registro.

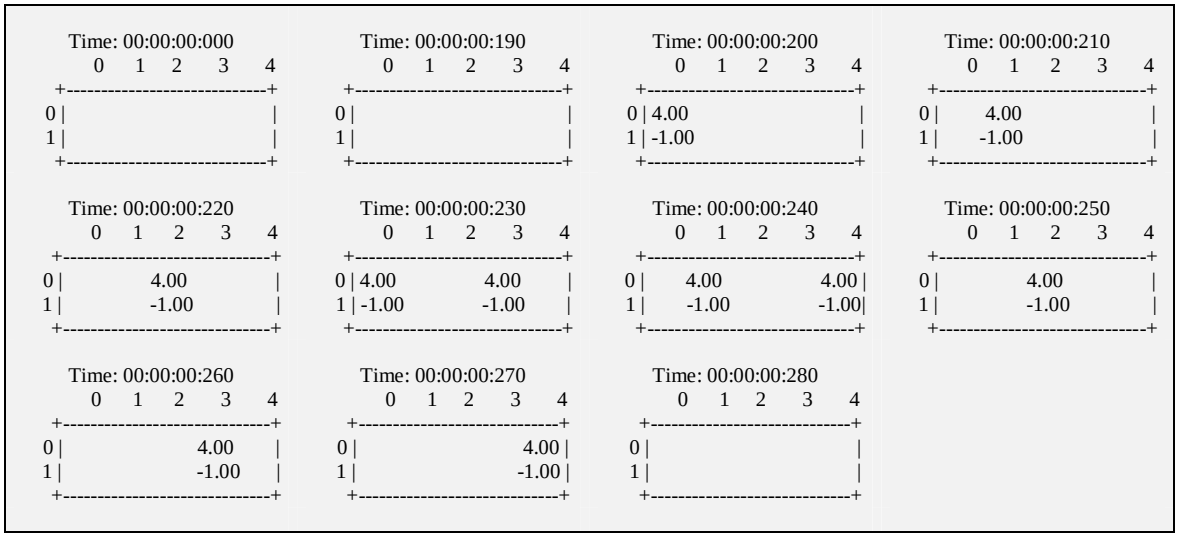
Como se observa en la figura del ejemplo, al finalizar el recorrido del tramo 5 los autos atraviesan al cruce 4.

Resultado de la simulación –Cruce3



Este cruce sólo posee una celda de entrada (celda 0) y una de salida (celda 1). Por lo tanto, las únicas acciones posibles son la entrada y salida de vehículos (no toma ninguna decisión de ruteo). Recibe los dos últimos autos ingresados a la simulación, por el tramo 3 en los tiempos 00:00:00:190 y 00:00:00:220 y los envía hacia el tramo 4 en el primer intento de entrada al mismo, ya que la primer celda del tramo se encuentra libre.

Resultado de la simulación – Tramo4



Los dos últimos autos ingresados a la simulación siguen el camino (*Tramo 3 - Tramo 4*), decisión tomada en el cruce1 (y luego no hay un cruce con capacidad de ruteo). Por lo tanto, ingresan a este tramo en los tiempos 00:00:00:200 y 00:00:00:230 (luego de dejar el cruce 1).

En el nivel 1, correspondiente al camino, se observa el valor –1 que indica que el tramo actual es el último que el auto recorrerá, es decir que el auto llegará a destino al ingresar al próximo cruce. Como se observa en la figura del ejemplo, al finalizar el recorrido del tramo 4 los autos atraviesan al cruce 4.

Resultado de la simulación –Cruce4

```
Time: 00:00:00:000
          0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00     4.00    |
3 |                   |
  +-------------------+

Time: 00:00:00:220
          0       1
  +-------------------+
0 |             4.00  |
1 |            -1.00  |
2 |  4.00     4.00    |
3 |                   |
  +-------------------+

Time: 00:00:00:230
          0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00     4.00    |
3 |                   |
  +-------------------+

Time: 00:00:00:250
          0       1
  +-------------------+
0 |  4.00             |
1 | -1.00             |
2 |  4.00     4.00    |
3 |                   |
  +-------------------+

Time: 00:00:00:260
          0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00     4.00    |
3 |                   |
  +-------------------+

Time: 00:00:00:280
          0       1
  +-------------------+
0 |  4.00     4.00    |
1 | -1.00    -1.00    |
2 |  4.00     4.00    |
3 |                   |
  +-------------------+

Time: 00:00:00:290
          0       1
  +-------------------+
0 |                   |
1 |                   |
2 |  4.00     4.00    |
3 |                   |
  +-------------------+
```

Finalmente, mostramos el resultado del cruce 4. Los autos que siguieron el camino (*Tramo 2* - *Tramo 5*) ingresan por la celda 1, en los tiempos 00:00:00:220 y 00:00:00:280. Mientras que, los autos que siguieron el camino (*Tramo 3* - *Tramo 4*) ingresan por la celda 0, en los tiempos 00:00:00:250 y 00:00:00:280. El cruce detecta que los autos han llegado a destino y los elimina de la simulación.

Sección 4.4: Comportamiento del conductor

En los modelos presentados hasta el momento, la velocidad de los vehículos se determina en forma aleatoria. En esta sección se modificará el modelo resultante de la incorporación de ruteo (dinámico), de manera que la velocidad de cada vehículo se calcule en base a la situación de contexto que se presenta, las características del conductor del vehículo y los atributos del auto.
Para la construcción de este modelo se ha asumido que:

- Se tiene una clasificación de tipos de conductores, en la cual se especifica las características de cada tipo (como se propuso en el inicio de la sección 4).
- Se tiene una clasificación de tipos de vehículos, para la cual se consideran los atributos que se estimen relevantes.
- Para cada vehículo, la situación de contexto está determinada por los vehículos que posee delante de él, conociendo a qué distancia se encuentra el vehículo más cercano y a que velocidad va.

Los modelos han sido construidos de tal forma que las clasificaciones de tipos de conductores y tipos de vehículos, así como sus atributos y la función que calcula la velocidad en base a estos parámetros puedan ser redefinidas sin afectar la definición del modelo de simulación.
Esto permite tratar en forma independiente dos aspectos: las características propias del problema y el modelo de simulación utilizado.

Definición de Modelos

En los modelos descriptos en la siguiente sección se incluye la siguiente información:

- Tipo de conductor
- Tipo de vehículo
- Distancia a la que se encuentra el auto más cercano (hacia adelante)
- Velocidad del auto que se encuentra más cercano (hacia adelante)

Se definen cuatro modelos distintos que sólo difieren en la forma en que obtienen y comunican la información anteriormente descripta.

Los modelos definidos pueden describirse brevemente de la siguiente forma:

	Nombre del modelo	Descripción
1	GAP	A cada tramo se le acopla un nuevo modelo DEVS que monitorea el estado de todo el tramo y le indica a cada celda todos los datos del auto más cercano que tiene delante. Llamaremos a este nuevo modelo 'GAP'.
2	Extensión de vecindario	Se modifica la definición del modelo correspondiente al tramo, extendiendo el vecindario de cada celda del tramo hasta una distancia de visibilidad standard. De esta forma, puede conocer a que distancia tiene el auto más cercano.
3	Propagación de información	Se mantiene la vecindad de una celda sólo con las celdas adyacentes y se agregan ports que van propagando la información del auto más cercano.
4	Estimación con información de congestión	No se usa información exacta, sino que se estima en base a algunos factores de contexto. Se actualiza la aceleración del auto en base a una función de la cantidad de carriles, cantidad de celdas del tramo e información de congestión. Como éste es un método aproximado, se agrega además una variable de ajuste dinámica que ayude corregir posibles desvíos en la estimación.

Calles y Cruces

La definición de las calles y los cruces utilizados en el modelo y especificados mediante el lenguaje ATLAS, es exactamente la misma que la especificada en los modelos anteriores.

Propuesta 1 - Modelo GAP

Este modelo ha sido definido tomando como base, el presentado en la sección 4.2 (ruteo dinámico).
Se agrega como característica de los vehículos:

- El tipo de conductor que lo conduce, y
- El tipo de vehículo (por ejemplo, auto, camioneta, colectivo, etc)

Por otro lado, se agrega información que permite conocer a qué distancia se encuentra el auto más cercano (hacia adelante) y que velocidad lleva. Para poder obtener esta información, a cada tramo se le acopla un nuevo modelo DEVS que monitorea el estado de todo el tramo y le indica a cada celda todos los datos del auto más cercano que tiene delante. Llamaremos a este nuevo modelo 'GAP'.
Los modelos de las calles y los cruces debieron ser modificados de la siguiente forma:

Calles de carril único

Un tramo ***t = (p1, p2, 1, a, dir, max, nro_tramo)*** de carril único se define como un modelo ***Cell_DEVS*** de una dimensión con demora de transporte, cuya estructura se presenta en la siguiente figura.

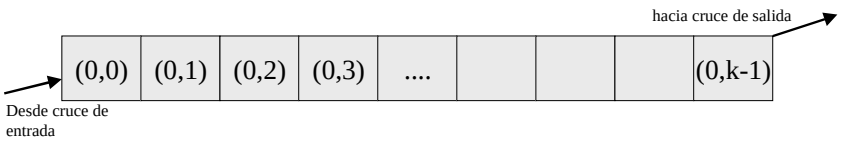


Figura 1– Tramo de carril único

Cada celda en este espacio se define como:

$$C_{0j}\ (nro\_tramo) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = <η, P^x, P^y>, donde

η = 3; y P^x y P^y donde
P^x = { (X_1, Record), (X_2, Record), (X_3, Record) }
P^y = { (Y_1, Record), (Y_2, Record), (Y_3, Record) }

X, Y ∈ N

S = (destino, camino, velocidad, tipo_vehiculo, tipo_conductor)

Donde

destino = { != 0 si hay un vehículo en la celda ; 0 sino.

Destino ∈ N: Destino del auto en la celda (cruce)

Camino = { $\{t_1.t_2 \ldots t_n\}$ donde ti ∈ N ∧ (∀i (∃ r ∈ Tramos/ Nro_tramo(r) = t_i)) si hay un vehículo en la celda ; 0 sino.

representa el camino que sigue el auto contenido en la celda

velocidad ∈ R: velocidad del auto que se encuentra en la celda

tipo_vehiculo ∈ N: tipo de vehículo

tipo_conductor ∈ N: tipo de conductor

Notas:

- destino = 0 ⇒ Camino = 0 (si no hay auto en la celda tampoco hay camino)
- destino = 0 ⇒ Velocidad = 0 (si no hay auto en la celda, la velocidad es cero)
- destino = 0 ⇒ tipo_vehiculo = 0 (si no hay auto en la celda, el tipo de vehículo es cero)
- destino = 0 ⇒ tipo_conductor = 0 (si no hay auto en la celda, el tipo de conductor es cero)

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

(0,-1)	(0,0)	(0,1)

Figura 2- Vecindario de la celda origen

delay = transport

d = ***Conversión_Demora (Velocidad (velocidad_actual,***
tipo_vehiculo,
tipo_conductor,
distancia_auto_cercano,
velocidad_auto_cercano,
Velocidad_maxima_tramo))

donde,

- ***velocidad:*** es una función que calcula la nueva velocidad del auto en base a la velocidad actual del mismo, al tipo de vehículo, el tipo de conductor, la distancia y

velocidad del auto más cercano hacia adelante y la velocidad máxima permitida en el mismo.

- ***Conversión_Demora:*** es una función detallada en el Apéndice E, que recibe como parámetro un valor natural que representa una velocidad (en km/h) y devuelve el tiempo (en segundos) que se debe demorar el auto en la celda para representar que avanza con esa velocidad.

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

τ: S x N $\rightarrow$ S se define de la siguiente manera,

FUNCION τ			
Nuevo estado	**Estado del vecindario**	**Nuevo estado de los ports**	**Nombre de la regla**
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Velocidad* = Velocidad (Velocidad(0,-1), Tipo_Vehiculo(0,-1), Tipo_Conductor(0,-1), Portvalue(x-g-distancia), Portvalue(x-g-velocidad), Max) *Tipo_Vehículo* = Tipo_Vehiculo(0,-1) *Tipo_Conductor* = Tipo_Conductor(0,-1)	Destino (0,-1) != 0 and Destino(0,0) = 0	/* Información al modelo GAP */ Send(1, y-g-hayauto) Send(Velocidad, y-g-velocidad)	Llega_Desde_Atrás
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_Vehículo* = 0 *Tipo_Conductor* = 0	Destino(0,0) != 0 and Destino(0,1) = 0	/* Info al modelo GAP */ Send(0, y-g-hayauto)	Sale_Hacia_Adelante
(0,0)	t /*en otro caso conserva estado*/		Default

El estado de una celda representa:

- La presencia (Destino(s) != 0) o ausencia (Destino(s) = 0) de un vehículo
- Las características del auto que se encuentra en la celda: Destino(s), Camino(s), Velocidad(s), Tipo_vehiculo(s) y Tipo_Conductor(s)

En un tramo de carril único sólo se permite que los vehículos avancen hacia la posición de adelante, siempre y cuando la misma esté vacía. De esta forma el vecindario necesario para establecer el nuevo estado de una celda está constituido por ella misma, la celda anterior y la siguiente.

Las demoras de transporte se utilizan para modelar las demoras provocadas por la aceleración de los automotores. El movimiento de los mismos se demora antes del siguiente movimiento a la próxima celda. Este valor es generado mediante el cálculo del tiempo necesario para alcanzar la velocidad especificada por la función 'Velocidad' que calcula la nueva velocidad que tomará el vehículo teniendo en cuenta la velocidad a la que venía el vehículo, el tipo de vehículo, el tipo de conductor, la distancia y velocidad del auto más cercano que tiene delante y la velocidad máxima permitida en el tramo.

El movimiento de los vehículos (especificado con la función τ) queda definido por 3 reglas:

- La regla *Llega_Desde_Atrás* representa el movimiento recto del vehículo desde la celda de atrás hacia la origen, siempre y cuando ésta se encuentre vacía.
- La regla *Sale_Hacia_Adelante* representa el movimiento recto del vehículo que se encuentra en la celda origen hacia la de adelante.
- La regla *Default* considera cualquier otro caso donde el estado de la celda no cambia.

Modelo acoplado

El modelo acoplado correspondiente al tramo *t = (p1, p2, 1, a, dir, max, nro_tramo)* se define como:

TC1(k, max, nro_tramo) = < Xlist, Ylist, I, X, Y, n, {t_1,...,t_n}, η, N, C, B, Z, select >

donde,

Ylist = { (0,r) / $0 \le r \le k-1$ }

Xlist = { (0,r) / $0 \le r \le k-1$ }

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,0), Natural>,
<$X_{\eta+2}$(0,0), Natural>,
<$X_{\eta+3}$(0,0), Real>,
<$X_{\eta+4}$(0,0), Natural>,
<$X_{\eta+5}$(0,0), Natural>,
<$X_{\eta+6}$(0,r), Real> con $0 \le r \le k-2$,
<$X_{\eta+7}$(0,r), Real> con $0 \le r \le k-2$,
<$X_{\eta+1}$(0,k-1), Binario>
}

P^y = {<$Y_{\eta+1}$(0,0), Binario>,
<$Y_{\eta+2}$(0,0), Binario >,
<$Y_{\eta+1}$(0,k-1), Natural>,
<$Y_{\eta+2}$(0,k-1), Binario>,
<$Y_{\eta+3}$(0,k-1), Natural>,
<$Y_{\eta+4}$(0,k-1), Natural>,
<$Y_{\eta+5}$(0,k-1), Natural>,
<$Y_{\eta+6}$(0,r), Binario> con $0 \le r \le k-1$,
<$Y_{\eta+7}$(0,r), Real> con $0 \le r \le k-1$,
<$Y_{\eta+8}$(0,0), Real >,
}

Los ports se denominarán de la siguiente forma:

PORTS		
Port	**Nombre**	**Comentario**
Celda (0,0) : Primer celda del tramo		
$X_{\eta+1}$(0,0)	x-c-destino	Este port se utiliza para informar de la salida de un auto desde el cruce hacia el tramo, informando su destino (portvalue($X_{\eta+1}$) = Destino).
$X_{\eta+2}$ (0,0)	x-c-camino	Este port se utiliza para informar el camino del auto que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+2}$) = Camino).
$X_{\eta+3}$(0,0)	*x-c-velocidad*	*Este port se utiliza para informar la velocidad del vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+3}$) = Velocidad).*
$X_{\eta+4}$(0,0)	*x-c-tipo_vehiculo*	*Este port se utiliza para informar el tipo de vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+4}$) = Tipo Vehículo).*

$X_{\eta+5}(0,0)$	*x-c-tipo_conductor*	*Este port se utiliza para informar el tipo de conductor del vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+5}$) = Tipo de conductor).*
$Y_{\eta+1}$ (0,0)	y-c-haylugar	Este port se utiliza para que el cruce pueda saber si hay lugar en el tramo para que un auto pueda salir de él (portvalue($Y_{\eta+1}$) = 0).
$Y_{\eta+2}$ (0,0)	y-r-EntraAuto	Este port informa al controlador de congestion la entrada de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+8}$ (0,0)	*y-c-veloc_maxima*	*Este port informa al cruce (celda de salida) la velocidad máxima permitida en el tramo. Toma el valor del parámetro especificado en el tramo: max.*
Celda (0,k-1) : Ultima celda del tramo		
$X_{\eta+1}$ (0,k-1)	x-c-haylugar	Este port permite saber si en el cruce hay lugar (portvalue($X_{\eta+1}$) = 0) para que avance un auto desde el tramo.
$Y_{\eta+1}$ (0,k-1)	y-c-destino	Este port informa al cruce de la presencia de un auto en el tramo que desea avanzar hacia él, informando su destino (portvalue($Y_{\eta+1}$) = Destino).
$Y_{\eta+2}$ (0,k-1)	y-r-SaleAuto	Este port informa al controlador de congestion la salida de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+3}$ (0,k-1)	y-c-camino	Este port se utiliza para informar el camino del auto que sale del tramo para ingresar al cruce (portvalue($Y_{\eta+3}$) = Camino).
$Y_{\eta+4}(0,k-1)$	*y-c-tipo_vehiculo*	*Este port se utiliza para informar el tipo de vehículo que sale hacia el cruce(portvalue($Y_{\eta+5}$) = Tipo Vehículo).*
$Y_{\eta+5}(0,k-1)$	*y-c-tipo_conductor*	*Este port se utiliza para informar el tipo de conductor del vehículo que sale hacia el cruce (portvalue($Y_{\eta+6}$) = Tipo de conductor).*
Comunicación con el modelo GAP.		
$X_{\eta+6}(0,r)$, $0 \le r \le k-2$	x-g-distancia	Este port se utiliza para que el modelo 'GAP' informe cuál es la distancia a la que se encuentra el auto de adelante más cercano.
$X_{\eta+7}(0,r)$, $0 \le r \le k-2$	x-g-velocidad	Este port se utiliza para que el modelo 'GAP' informe cuál es la velocidad del auto de adelante más cercano.
$<Y_{\eta+6}(0,r)$, $0 \le r \le k-1$	y-g-hayauto	Este port informa al modelo GAP la presencia de un auto en la celda (portvalue($Y_{\eta+8}$) = 1).
$<Y_{\eta+7}(0,r)$, $0 \le r \le k-1$	y-g-velocidad	Este port informa al modelo GAP la velocidad a la cual va el auto que se encuentra en la celda.

Nota: Los ports descriptos en letra *cursiva*, son los agregados por este nuevo modelo

$\boldsymbol{X} \in R$
$\boldsymbol{Y} \in R$
n = 1
$\mathbf{t_1}$ = k
$\boldsymbol{\eta}$ = 3
N = { (0,-1), (0,0), (0,1) }
C = { C_{ij} / i = 0 $\wedge$ j $\in$ [0, k-1] }, donde cada C_{ij} es un componente Cell-DEVS con demora. La especificación de cada celda ha sido descripta antes de introducir el modelo acoplado.
B = { (0,r) con $0 \le r \le k-1$ }
Z se construye siguiendo la definición dada por el formalismo Cell_DEVS, instanciada con el vecindario de este espacio.
select = { (0,1), (0,0), (0,-1) }

La especificación de este modelo representa el movimiento de los vehículos en calles de carril único. Cada tramo se extiende entre un cruce de origen y otro de destino, por lo tanto se definen los ports que acoplan estos modelos y permiten el avance de tráfico de uno al otro.
La interfaz la conforman todas las celdas. Las celdas (0,0) y (0,k-1) intercambian información con los modelos de los cruces correspondientes y el controlador de congestión que lo monitorea. Las celdas (0, r) con $0 \leq r \leq k-1$ intercambian información con el modelo GAP para conocer la velocidad y posición del auto más cercano.

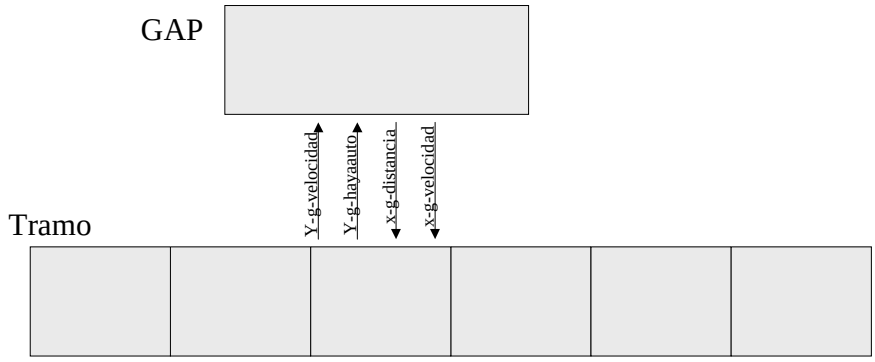


Figura 3: Comunicación del tramo con el modelo GAP

Celdas borde
Celda (0,0)
El comportamiento para las celdas borde es distinto al definido anteriormente. Para la celda (0,0) se define el siguiente vecindario y comportamiento:

$\eta = 2$
N = { (0,0), (0,1) }

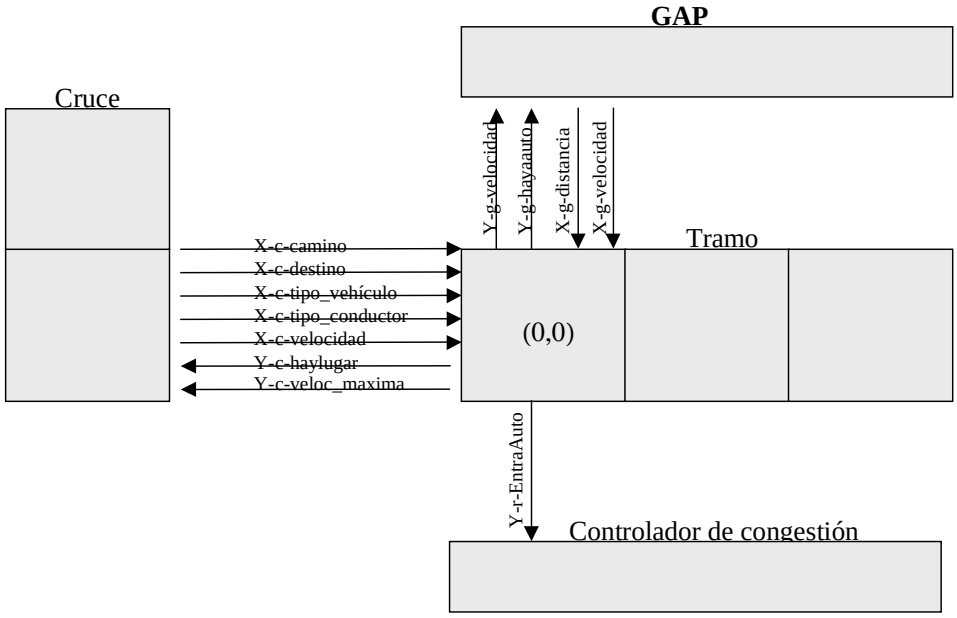


Figura 4- Vecindario y acoplamiento de la celda (0,0)

La función τ para esta celda se define como:

FUNCION τ			
Nuevo Estado	**Estado del vecindario**	**Nuevo estado de los ports**	**Nombre de la regla**
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_Vehículo* = 0 *Tipo_Conductor* = 0	(Destino(0,0) != 0 and Destino(0,1) = 0)	/* Info al GAP */ Send(0, y-g-hayauto)	Sale_Hacia_Adelante
Destino = portvalue(x-c-destino) *Camino* = portvalue(x-c-camino) *Velocidad* = Velocidad (portvalue(x-g-velocidad), portvalue(x-c-tipo_Vehiculo), portvalue(x-c-tipo_Conductor), portvalue(x-g-distancia), portvalue(x-g-velocidad), Max) *Tipo_Vehículo* = portvalue(x-c-tipo_Vehiculo) *Tipo_Conductor* = portvalue(x-c-tipo_Conductor)	(portvalue(x-c-destino) != 0 and Destino(0,0) = 0)	/* Info de congestion */ Send(1, y-r-EntraAuto) /* Info al GAP */ Send(1, y-g-hayauto) Send(Velocidad, y-g-velocidad)	Llega_Desde_Cruce
(0,0)	t /*en otro caso conserva estado*/		Default

Los demás parámetros del modelo para esta celda no cambian.

La celda (0,0) tiene el vecindario formado por ella y la celda de adelante, además se encuentra acoplada a la celda del cruce desde la que recibe los vehículos, al controlador de congestión y al modelo 'GAP'. Esto se puede ver en la figura 4
Para conocer el estado de la celda del cruce se utiliza el port x-c-destino, siempre que sea distinto de 0 representa que existe un vehículo que saldrá del cruce hacia la celda (0,0), y por consiguiente se informará el destino, camino, tipo del vehículo, tipo de conductor y velocidad actual del auto contenida en ella.
También, el tramo informará al controlador de congestión de la entrada de un auto al tramo.
Por lo tanto, la regla Llega_Desde_Cruce representa el movimiento de avance de un vehículo que se encuentra en el cruce (portvalue(x-c-destino) != 0) hacia la celda (0,0). Las otras 2 reglas son las mismas que fueron definidas para el resto de las celdas.
La celda (0,0) envía su estado hacia el cruce a través del port y-c-haylugar, es decir 0 si está vacía y 1 en otro caso.

Celda (0,k-1)

Para la celda (0,k-1) también se define un vecindario y comportamiento diferente al resto. Aquí:

$\eta = 2$

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

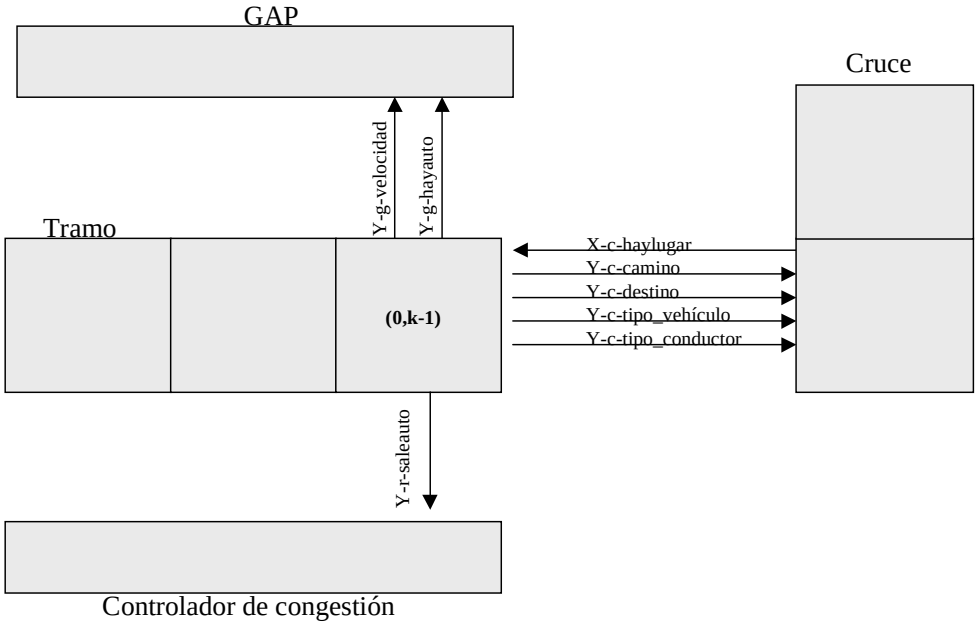


Figura 5- Vecindario y acoplamiento de la celda (0,k-1)

La función τ y el tipo de demora (delay) para esta celda se define como:

FUNCION τ y tipo de demora (delay)				
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**	**Delay**	**Nombre de la regla**
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_Vehículo* = 0 *Tipo_Conductor* = 0	(Destino(0,0) != 0 and portvalue(x-c-haylugar) = 0)	/* Info al cruce */ Send (Destino(0,0), y-c-Destino) Send (Camino(0,0), y-c-Camino) Send (Tipo_Vehiculo(0,0), y-c-tipo_vehiculo) Send (Tipo_conductor(0,0), y-c-tipo_conductor) /* Info de congestion */ Send(1, y-r-SaleAuto) /* Info al GAP */ Send(0, y-g-hayauto)	Inercial	Sale_Hacia_Cruce
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Velocidad* = Velocidad_Ingreso_Cruce *Tipo_Vehículo* = Tipo_Vehiculo(0,-1) *Tipo_Conductor* = Tipo_Conductor(0,-1) donde Velocidad_Ingreso_Cruce es un parámetro de la simulación.	(Destino(0,-1) != 0 and Destino(0,0) = 0)	/* Info al cruce */ Send(0, y-c-destino) /* Info al GAP */ Send(1, y-g-hayauto) Send(Velocidad_Ingreso_Cruce, y-g-velocidad)	Transport	Llega_Desde_Atrás
(0,0)	t /*en otro caso conserva estado */	/* Info al cruce */ send(0, y-c-destino)	Transport	Default

Los demás parámetros del modelo para esta celda no cambian.

La celda (0,k-1) tiene el vecindario formado por ella y la celda de atrás, además se encuentra acoplada con 1 celda del cruce, al controlador de congestión y al modelo GAP. Esto se puede ver en la figura 5.
La celda se comporta de manera análoga a los modelos definidos anteriormente.
Las reglas de movimiento quedan definidas de la siguiente forma:

- La regla *Sale_Hacia_Cruce* representa el avance del vehículo desde la celda (0,k-1) hacia la de adelante (dentro del cruce), siempre y cuando ésta y su anterior se encuentren vacías (portvalue(x-c-haylugar) = 0). No se envía la velocidad del auto, ya que la velocidad de ingreso y de circulación del cruce se asume que será igual para todos los vehículos y se especifican mediante parámetros de la simulación.
- La regla *Llega_Desde_Atrás* representa el movimiento recto del vehículo desde la celda de atrás hacia la celda de salida del tramo, siempre y cuando ésta se encuentre vacía. La velocidad que tomará el vehículo cuando se acerca al cruce se representará mediante un valor constante definido como parámetro de la simulación, ya que se asume que al llegar al mismo los vehículos circulan a una velocidad dentro de un rango acotado, que permita observar la presencia de otros vehículos dentro del cruce.
- La regla *Default* considera cualquier otro caso donde el estado de la celda no cambia.

La celda (0,k-1) se modela con los 2 tipos de demora (inercial y transporte) para representar diferentes comportamientos. Por un lado, cuando un vehículo quiere avanzar hacia la celda (0,k-1), regla *Llega_Desde_Atrás*, se utiliza demora de transporte, pues este movimiento no varía del realizado en cualquier otra celda del tramo a pesar que (0,k-1) esté en el borde. Pero para ingresar al cruce, es decir abandonar la celda (0,k-1), se modela con demora inercial porque representa el comportamiento de los vehículos al llegar a las esquinas y en general este tipo de maniobra se realiza con más precaución que el avance sobre el resto de la calle. Los automotores avanzan sobre el cruce si durante un cierto tiempo (demora inercial) tienen el camino libre, caso contrario esperan a que pasen los vehículos que ya se encuentran en el mismo. De esta forma los autos que circulan dentro del cruce tienen mayor probabilidad de avanzar que aquellos que desean ingresar y estos últimos no lo hacen si no tienen el camino libre por el tiempo que les lleva realizar la maniobra.

Los modelos correspondientes a los tramos de 2, 3, 4 y más de 4 carriles sufren modificaciones análogas a las del caso de un carril:

- Cambia la definición del estado.
- Se conectan al modelo GAP para informar su estado. En este caso el modelo GAP debe procesar mayor cantidad de información y su función será mas compleja. Esta información puede ser también utilizada para la decisión de cambiar de carril.

La definición de los modelos correspondientes a estos casos no será presentada en esta tesis y queda como propuesta para trabajos futuros.

Definición del modelo que representa a los cruces

Con el objetivo de evitar repetir la explicación descripta en secciones anteriores, se mostrarán sólo las modificaciones necesarias al modelo que representa a los cruces para incorporar el modelado de la velocidad a la cual circulan los vehículos.
El modelo base en el cual se describen las modificaciones es el presentado en la sección de Ruteo dinámico.

La funcionalidad agregada fue la siguiente:
Un vehículo posee el siguiente comportamiento al llegar a un cruce:

- Cuando se acerca al cruce disminuye su velocidad significativamente.
- Una vez dentro del cruce, aumenta su velocidad levemente ya que circula con precaución porque no conoce el estado de la salida del mismo.
- Una vez que decide salir, actualiza su velocidad de acuerdo al estado del tramo de salida.

Este comportamiento se ha modelado de la siguiente manera:

- Se definieron dos parámetros de la simulación:
  - *Velocidad_Ingreso_Cruce:* con esta velocidad ingresarán los autos a los cruces. La idea es que los autos ingresen con prudencia a los cruces, por lo tanto, este parámetro debería tener un valor bajo.
  - *Velocidad_dentro_cruce:* Como el rango de velocidades que pueden tomar los vehículos dentro de un cruce es acotado, decidimos modelar con velocidad constante el paso de los vehículos por el cruce. Esta velocidad constante se especifica en este parámetro, el cual debería ser mayor a 'Velocidad_Ingreso_Cruce' para representar la aceleración de los vehículos una vez que están dentro del cruce.

- Al salir del cruce se calcula la velocidad a tomar en base a las características del vehículo y el estado del tramo de salida (la distancia y velocidad del auto más cercano). Para ello, se conecta al modelo GAP, el cual le brinda esta información.

Se modifica el estado del modelo de la siguiente forma:

S = (Destino, Camino, Nro_Cruce, Nro_Tramo, Tipo_Vehiculo, Tipo_Conductor)
Donde

$$\textbf{\textit{destino}} = \begin{cases} \text{!=0 si hay un vehículo en la celda} \\ \text{0 sino.} \end{cases}$$

Destino $\in N$*:* Destino del auto en la celda (cruce)

$$\textbf{\textit{Camino}} = \begin{cases} \{t_1.t_2 \ldots t_n\} \text{donde } ti \in N \wedge (\forall i\, (\exists\, r \in \text{Tramos/ Nro\_tramo}(r) = t_i)) \\ \qquad \text{si hay un vehículo en la celda} \\ 0 \text{ sino.} \end{cases}$$

representa el camino que sigue el auto contenido en la celda

Nro_cruce $\in N$*:* Identificador del cruce (parámetro del modelo)

Nro_tramo $\in N$*:* identificador del tramo que se conecta a la celda de salida del cruce

tipo_vehiculo $\in N$*:* tipo de vehículo

tipo_conductor $\in$ *N:* tipo de conductor

Notas:

- destino = 0 $\Rightarrow$ Camino = 0 (si no hay auto en la celda tampoco hay camino)
- destino = 0 $\Rightarrow$ tipo_vehiculo = 0 (si no hay auto en la celda, el tipo de vehículo es cero)
- destino = 0 $\Rightarrow$ tipo_conductor = 0 (si no hay auto en la celda, el tipo de conductor es cero)
- La velocidad del auto no se incluye dentro del estado, ya que es constante y está determinada por el parámetro 'Velocidad_dentro_cruce'.

El estado de una celda representa:

- La presencia o ausencia de un vehículo.
- El destino del auto dentro de ella.
- El camino que seguirá el auto contenido en la celda.
- El identificador único del cruce.
- Sólo para las celdas de salida, se especifica la identificación del tramo al cual está conectada.
- El tipo de conductor del auto dentro del cruce.
- El tipo de vehículo del auto dentro del cruce.

d = Conversión_Demora(VELOCIDAD_DENTRO_CRUCE)
donde VELOCIDAD_DENTRO_CRUCE es un parámetro del sistema

Modelo acoplado

Al modelo presentado en la sección 4.2 (Ruteo dinámico), se han agregado los siguientes ports:

PORTS		
Port	**Nombre**	**Comentario**
Celdas de entrada		
$X_{\eta+4}(0, i)$, $i \in$ In	x-t-tipo_vehiculo	Este port se utiliza para conocer el tipo de vehículo que ingresa desde el tramo hacia el cruce (portvalue($X_{\eta+4}$) = Tipo Vehículo).
$X_{\eta+5}(0, i)$, $i \in$ In	x-t-tipo_conductor	Este port se utiliza para conocer el tipo de conductor del vehículo que ingresa desde el tramo hacia el cruce (portvalue($X_{\eta+5}$) = Tipo de conductor).
Celdas de salida		
$X_{\eta+7}(0,i)$, $i \in$ Out	x-g-distancia	Este port se utiliza para conocer la distancia a la que se encuentra el auto más cercano (de adelante) en el tramo de salida.
$X_{\eta+8}(0,i)$, $i \in$ Out	x-g-velocidad	Este port se utiliza para conocer la velocidad que tiene el auto más cercano (de adelante) que se encuentra en el tramo de salida
$X_{\eta+9}(0,i)$, $i \in$ Out	x-t-veloc_maxima	Este port se utiliza para conocer la velocidad máxima a la que se puede ir en el tramo de salida.
$Y_{\eta+4}(0, i)$, $i \in$ Out	y-t-tipo_vehiculo	Este port se utiliza para informar el tipo de vehículo que sale hacia el tramo(portvalue($Y_{\eta+4}$) = Tipo Vehículo).
$Y_{\eta+5}(0, i)$, $i \in$ Out	y-t-tipo_conductor	Este port se utiliza para informar el tipo de conductor del vehículo que sale hacia el tramo (portvalue($Y_{\eta+5}$) = Tipo de conductor).
$Y_{\eta+6}(0, i)$, $i \in$ Out	y-t-velocidad	Este port se utiliza para informar la velocidad del vehículo que sale hacia el tramo (portvalue($Y_{\eta+6}$) = Velocidad).

Las reglas correspondientes a las celdas de entrada y salida del cruce, se modifican de la siguiente forma:

Celdas de ingreso al cruce

Las celdas de ingreso al cruce son: ***{ (0,i) / i ∈ In }***

FUNCION τ		
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = nro_Tramo(0,0) *Tipo_vehiculo* = Tipo_vehiculo(0,-1) *Tipo_conductor* = Tipo_Conductor(0,-1)	Destino(0,0) = 0 and Destino(0,-1) != 0 */* Ingresa a la celda un auto que ya estaba dentro del cruce, conservando el mismo camino */*	Send(1, y-t-haylugar)
Destino = portvalue(x-t-destino) *Camino* = portvalue(x-t-Camino) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0) *Tipo_vehiculo* = portvalue(x-t-tipo_vehiculo) *Tipo_conductor* = portvalue(x-t-tipo_conductor)	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and portvalue(x-t-destino) != nro_cruce(0,0) and ¬Hay_Congestion(portvalue(x-c-congestion $_{Proximo\_Tramo(Camino(0,-1))}$) , nro_Tramo(0,0))) */* Entra al cruce un auto que todavía no ha llegado a destino, conservando el mismo camino con el que venía */*	Send(1, y-t-haylugar)
Destino = portvalue(x-t-destino) *Camino* = Nuevo_Camino(Nro_Cruce(0,0), Portvalue(x-t-destino), Portvalue(x-t-camino)) *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0) *Tipo_vehiculo* = portvalue(x-t-tipo_vehiculo) *Tipo_conductor* = portvalue(x-t-tipo_conductor)	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and portvalue(x-t-destino) != nro_cruce(0,0) and Hay_Congestion(portvalue(x-c-congestion $_{Proximo\_Tramo(Portvalue(x-t-camino))}$) , nro_Tramo(0,0))) */* Entra al cruce un auto que todavía no ha llegado a destino, cambiando el camino por encontrarse con congestión */*	Send(1, y-t-haylugar)
Destino = 0 *Camino*= 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0) *Tipo_vehiculo* = 0 *Tipo_conductor* = 0	Destino(0,0) = 0 and Destino(0,-1) = 0 and Portvalue(x-t-destino) != 0 and portvalue(x-t-destino) = nro_cruce(0,0) */* Elimina el auto del modelo ya que ha llegado a destino */*	Send(0, y-t-haylugar)
Destino = 0 *Camino* = 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0) *Tipo_vehiculo* = 0 *Tipo_conductor* = 0	Destino(0,0) != 0 and Destino(0,1) = 0 and Destino(0,-1) = 0 */* No hay un auto con prioridad dentro del cruce */*	Send(0, y-t-haylugar)
Destino = 0 *Camino* = 0 *Nro_cruce* = nro_cruce(0,0) *Nro_tramo* = Nro_Tramo(0,0) *Tipo_vehiculo* = 0 *Tipo_conductor* = 0	Destino(0,0) != 0 and Destino(0,1) = 0 and Destino(0,-1) != 0 */* Hay un auto con prioridad dentro del cruce */*	Send(1, y-t-haylugar)

(0,0)	T */* En otro caso la celda conserva el estado */*	-

Las celdas de ingreso no varían su comportamiento, sólo se agrega la transmisión de la información sobre el tipo de conductor y el tipo de vehículo.

Celdas de salida del cruce

Las celdas de salida del cruce son: ***{ (0,i) / i ∈ Out }***

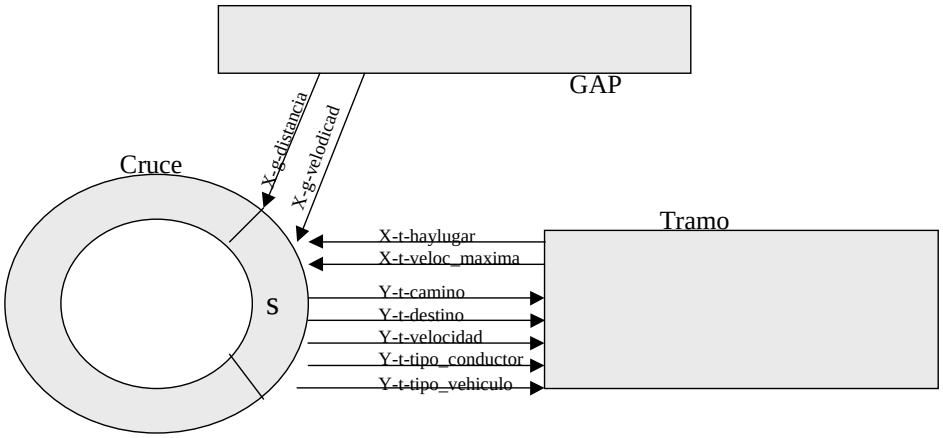


Figura 6- Acoplamiento de las celdas de salida de un cruce (s es una celda de salida)

	FUNCION τ	
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**
Destino = Destino(0,-1) *Camino*= Camino(0,-1) *nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Tipo_vehiculo* = Tipo_vehiculo(0,-1) *Tipo_conductor* = Tipo_conductor(0,-1)	Destino(0,0) = 0 and Destino(0,-1) != 0 and ((portvalue(x-t-haylugar) = 1) or (portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) != Nro_Tramo(0,0))) */* Llega auto que permanecerá dentro del cruce */*	Send(0, y-t-destino)
Destino = 0 *Camino*= 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Tipo_vehiculo* = 0 *Tipo_conductor* = 0	Destino(0,0) = 0 and Destino(0,-1) != 0 and Portvalue(x-t-haylugar) = 0 and Proximo_Tramo(Camino(0,-1)) = Nro_Tramo(0,0) */* Llega auto que abandonará el cruce */*	Send(Destino(0,-1), y-t-destino) Send(Cola_Camino(Camino(0,-1)), y-t-camino) Send(Tipo_vehiculo(0,-1), y-t-tipo_vehiculo) Send(Tipo_conductor(0,-1), y-t-tipo_conductor) Send(Velocidad(Velocidad_Cruce,

		Tipo_Vehiculo(0,-1), Tipo_Conductor(0,-1), portvalue(x-g-distancia), portvalue(x-g-velocidad), portvalue(x-t-velocidad_max)), y-t-velocidad)
Destino = 0 *Camino*= 0 *Nro_cruce*= nro_cruce(0,0) *Nro_tramo*= nro_tramo(0,0) *Tipo_vehiculo* = 0 *Tipo_conductor* = 0	Destino(0,0) != 0 and Destino(0,1) = 0 /* Sigue en el cruce */	Send(0, y-t-destino)
(0,0)	t /*En otro caso la celda conserva el estado */	Send(0, y-t-destino)

A este tipo de celdas se le agregan las características del vehículo (tipo de vehículo y tipo de conductor.
Por otra parte, la función 'Velocidad' calcula la aceleración que tendrá el vehículo al salir del cruce, teniendo en cuenta el tipo de vehículo, el tipo de conductor, la distancia y velocidad del auto más cercano, la velocidad máxima que permite el tramo al que está entrando y la velocidad actual del vehículo. Con esta velocidad ingresará el vehículo al tramo.

Definición del Modelo GAP

Se define el modelo 'GAP' como un modelo atómico, donde el parámetro k es la cantidad de celdas que tiene el tramo que este modelo monitorea.

$$M(k) = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

Donde:
$\boldsymbol{X}$ = {<X_i, Binario> con $0 \le i \le k-1$, <X_j, Real> con $k \le j \le 2k-1$ }

$\boldsymbol{Y}$ = {<Y_i, Natural> con $0 \le i \le k-2$, <Y_j, Real> con $k \le j \le 2k-2$, <Y_{2k-1}, Real>, <Y_{2k}, Real> }

Los ports se denominarán de la siguiente forma:

PORTS		
Port	**Nombre**	**Comentario**
X_i con $0 \le i \le k-1$	x-t-hayauto	Este port se utiliza para informarle al modelo GAP si en la celda i del tramo hay un auto.
X_j con $k \le j \le 2k-1$	x-t-velocidad	Este port se utiliza para informarle al modelo GAP la velocidad del auto que está en la celda j – k del tramo.
Y_i con $0 \le i \le k-2$	y-t-distancia	Este port le informa a la celda i del tramo, cual es la distancia a la que se encuentra el auto más cercano (hacia adelante).
Y_j con $k \le j \le 2k-2$	y-t-velocidad	Este port le informa a la celda j – k del tramo, cuál es la velocidad del auto más cercano (hacia adelante).
Y_{2k-1}	y-c-distancia	Este port le informa al cruce, cual es la distancia a la que se encuentra el auto más cercano (hacia adelante) dentro del tramo.
Y_{2k}	y-c-velocidad	Este port le informa al cruce, cuál es la velocidad del auto más cercano (hacia adelante) dentro del tramo.

S = (Hay_auto[0, k-1], Velocidad[0, k-1]) donde Hay_Auto y Velocidad son dos vectores de k posiciones.

$$\textit{Hay\_Auto(i)} = \begin{cases} 1 \text{ si hay un vehículo en la celda i del tramo;} \\ 0 \text{ sino.} \end{cases}$$

Velocidad(i) es igual a la velocidad del auto que se encuentra en la celda i del tramo.

Variable descriptiva del modelo:
Celda_Cambia: $\in$ N y $0 \leq$ Celda_Cambia $\leq$ k-1

δ_{int} = Id (Función identidad ya que no se producen cambios de estado por transiciones internas)

δ_{ext}
Cuando recibo en el port x-t-hayauto(i)
Hay_Auto(i) = x-t-hayauto(i)
Celda_Cambia = i

Cuando recibo en el port x-t-velocidad(i)
Velocidad(i) = x-t-velocidad(i)
Celda_Cambia = i

λ define como:

Devuelve para max(Celda_Cambia – Visibilidad, 0) $\leq$ j < Celda_Cambia
por el port y-t-distancia(j) : calcula_dis(j)
por el port y-t-velocidad(j) : Velocidad[j + calcula_dis(j)]

Si Celda_Cambia $\leq$ Visibilidad (indica que se le está enviando información al cruce)
Devuelve
por el port y-c-distancia : calcula_dis(0) + 1
por el port y-c-velocidad : Velocidad[1 + calcula_dis(0)]

Ta = 0

Acoplamiento del modelo GAP con los tramos y cruces

Acoplamiento con los tramos:
El modelo GAP se conecta con cada una de las celdas de un único tramo.
Mediante la comunicación de los ports (Tramo.y-g-hayauto(i) / GAP.x-t-hayauto(i)) e (Tramo.y-g-velocidad(i) / GAP.x-t-velocidad(i)) se le informa al modelo gap, para cada una de las celdas, si se encuentra ocupada y cuál es la velocidad del auto contenido en ella.
Mediante los ports (GAP.y-t-distancia(i) / Tramo.x-g-distancia(i)) e (GAP.y-t-velocidad(i) / Tramo.x-g-velocidad(i)) se informa a cada una de las celdas a qué distancia se encuentra el auto más cercano (hacia adelante) y cuál es su velocidad.

Acoplamiento con el cruce:
El modelo GAP se conecta con el cruce por el cual ingresan autos al tramo.
Mediante los ports (GAP.y-c-distancia / Cruce.x-g-distancia) y (GAP.y-c-velocidad / Cruce.x-g-velocidad)) se informa a cada una de las celdas a qué distancia se encuentra el auto más cercano (hacia adelante) y cuál es su velocidad.

Por lo tanto, el acoplamiento se define de la siguiente forma:
Sea k la cantidad de celdas del tramo
Ports de salida tramo – ingreso modelo GAP

- $Y_{\eta+6}$ (0, r) (Tramo) → X_r (GAP) con $0 \leq r \leq k-1$, informando si hay un auto en la celda r.
- $Y_{\eta+7}$ (0, r) (Tramo) → X_{k+r} (GAP) con $0 \leq r \leq k-1$, informando la velocidad del auto en la celda r.

Ports de salida modelo GAP – ingreso tramo

- Y_r (GAP) → $X_{\eta+6}$ (0, r) (Tramo) con $0 \leq r \leq k-2$, informando la distancia del auto más cercano a la celda r.
- Y_{k+r} (GAP) → $X_{\eta+7}$ (0, r) (Tramo) con $0 \leq r \leq k-2$, informando la velocidad del auto más cercano a la celda r.

Ports de salida modelo GAP – ingreso cruce

- Y_{2k-1} (GAP) → $X_{\eta+7}$ (Cruce), informando la distancia del auto más cercano a la celda dentro del tramo de salida del cruce (celdas de salida del cruce).
- Y_{2k} (GAP) → $X_{\eta+8}$ (Cruce), informando la velocidad del auto más cercano a la celda dentro del tramo de salida del cruce (celdas de salida del cruce).

Acoplamiento de los tramos con los cruces

Nota: Sólo se especificarán los nuevos acoplamientos:

Ports de salida tramo – ingreso cruce

- $Y_{\eta+4}$ (0,k-1) (y-c-tipo_vehiculo) → $X_{\eta+4}$ (Celda de entrada: x-t-tipo_vehiculo) Informando el tipo del vehículo que sale desde el tramo hacia el cruce.
- $Y_{\eta+5}$ (0,k-1) (y-c-tipo_conductor) → $X_{\eta+5}$ (Celda de entrada: x-t-tipo_conductor) Informando el tipo de conductor del auto que sale desde el tramo hacia el cruce.
- $Y_{\eta+8}$ (0,0) (y-c-veloc_maxima) → $X_{\eta+9}$ (Celda de salida: x-t-veloc_maxima) Informando la velocidad máxima permitida en el tramo hacia el cual va a salir el auto desde el cruce.

Ports de salida cruce – ingreso tramo

- $Y_{\eta+6}$ (Celda de salida: y-t-velocidad) → $X_{\eta+3}$ (0,0) (x-c-velocidad) Informando la velocidad del auto que ingresa desde el cruce hacia el tramo.
- $Y_{\eta+4}$ (Celda de salida: y-t-tipo_vehículo) → $X_{\eta+4}$ (0,0) (x-c-tipo_vehiculo) Informando el tipo de vehículo que ingresa desde el cruce hacia el tramo.
- $Y_{\eta+5}$ (Celda de salida: y-t-tipo_conductor) → $X_{\eta+5}$ (0,0) (x-c-tipo_conductor) Informando el tipo de conductor del auto que ingresa desde el cruce hacia el tramo.

Propuesta 2 - Extensión de vecindario

Este modelo ha sido definido tomando como base, el presentado en la sección 4.2 (ruteo dinámico). Se agrega como característica de los vehículos:

- El tipo de conductor que lo conduce, y
- El tipo de vehículo (por ejemplo, auto, camioneta, colectivo, etc)

Por otro lado, se agrega información que permite conocer a qué distancia se encuentra el auto más cercano (hacia adelante) y que velocidad lleva. Para poder obtener esta información, se extiende el vecindario de la celda en los modelos que corresponden a los tramos. El auto, ahora ve varias celdas hacia adelante, la cantidad de celdas depende de un parámetro de la simulación que representa la visibilidad que tiene el vehículo.
Los modelos de las calles y los cruces debieron ser modificados de la siguiente forma:

Calles de carril único

Un tramo ***t = (p1, p2, 1, a, dir, max, nro_tramo)*** de carril único se define como un modelo ***Cell_DEVS*** de una dimensión con demora de transporte, cuya estructura se presenta en la figura 1.

Cada celda en este espacio se define como:

$$C_{0j}\ (nro\_tramo) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = $<\eta, P^x, P^y>$, donde

η = v+2; y P^x y P^y donde
P^x = { (X_1, Record), …, (X_{v+2}, Record) }
P^y = { (Y_1, Record), …, (Y_{v+2}, Record) }.

X, Y $\in$ N

S = (destino, camino, velocidad, tipo_vehiculo, tipo_conductor)

Donde

destino = { != 0 si hay un vehículo en la celda ; 0 sino.

Destino $\in$ ***N:*** Destino del auto en la celda (cruce)

Camino = { $\{t_1.t_2 \ldots t_n\}$ donde $t_i \in N \wedge (\forall i\ (\exists\ r \in$ Tramos/ Nro_tramo(r) = t_i)) si hay un vehículo en la celda ; 0 sino.

representa el camino que sigue el auto contenido en la celda

velocidad $\in$ ***R:*** velocidad del auto que se encuentra en la celda

tipo_vehiculo $\in$ ***N:*** tipo de vehículo

tipo_conductor ∈ N: tipo de conductor

Notas:

- destino = 0 ⇒ Camino = 0 (si no hay auto en la celda tampoco hay camino)
- destino = 0 ⇒ Velocidad = 0 (si no hay auto en la celda, la velocidad es cero)
- destino = 0 ⇒ tipo_vehiculo = 0 (si no hay auto en la celda, el tipo de vehículo es cero)
- destino = 0 ⇒ tipo_conductor = 0 (si no hay auto en la celda, el tipo de conductor es cero)

N = { (0,-1), (0,0), (0,i) con $1 \leq i \leq v$}

(0,-1)	(0,0)	(0,1)	....	(0,v)	....

Figura 7- Vecindario de la celda origen

delay = transport

d = ***Conversión_Demora (Velocidad (velocidad_actual,***
tipo_vehiculo,
tipo_conductor,
distancia_auto_cercano,
velocidad_auto_cercano,
Velocidad_maxima_tramo))

donde,

- ***Velocidad:*** es una función que calcula la nueva velocidad del auto en base a la velocidad actual del mismo, al tipo de vehículo, el tipo de conductor, la distancia y velocidad del auto más cercano hacia adelante y la velocidad máxima permitida en el mismo.
- ***Conversión_Demora:*** es una función detallada en el Apéndice E, que recibe como parámetro un valor natural que representa una velocidad (en km/h) y devuelve el tiempo (en segundos) que se debe demorar el auto en la celda para representar que avanza con esa velocidad.

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

τ: S x N → S se define de la siguiente manera,

FUNCION τ		
Nuevo estado	**Estado del vecindario**	**Nombre de la regla**
Sea d = Distancia ({ Destino(0, i), 1≤i≤v }) que indica en que celda se encuentra el auto más cercano que tiene delante. El nuevo estado es: *Destino* = Destino(0,-1) *Camino* = Camino(0,-1) *Velocidad* = Velocidad (Velocidad(0,-1), Tipo_Vehiculo(0,-1), Tipo_Conductor(0,-1), d, Velocidad(0,d), Max) *Tipo_Vehículo* = Tipo_Vehiculo(0,-1) *Tipo_Conductor* = Tipo_Conductor(0,-1)	Destino (0,-1) != 0 and Destino(0,0) = 0	Llega_Desde_Atrás
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_Vehículo* = 0 *Tipo_Conductor* = 0 *Pendiente* = Pendiente(0,0)	Destino(0,0) != 0 and Destino(0,1) = 0	Sale_Hacia_Adelante
(0,0)	t /*en otro caso conserva estado*/	Default

Para conocer cuál es el auto más cercano a la celda se usa la función 'Distancia' a la cual se le pasa como parámetro una lista con los estados (indica sólo el destino de los autos contenidas en ellas, si el destino es igual a 0 quiere decir que la celda está libre) de todas las celdas de la vecindad que tiene delante, y devuelve el nro de la primer celda ocupada.
La velocidad se obtiene directamente del port de la primer celda ocupada.

Modelo acoplado

El modelo acoplado correspondiente al tramo *t = (p1, p2, 1, a, dir, max, nro_tramo)* se define como:

TC1(k, max, nro_tramo) = < Xlist, Ylist, I, X, Y, n, {t_1,...,t_n}, η, N, C, B, Z, select >

donde,

Ylist = { (0,0), (0,k-1) }

Xlist = { (0,0), (0,k-1) }

I = <P^x, P^y>

P^x = {<$X_{\eta+1}$(0,0), Natural>,
<$X_{\eta+2}$(0,0), Natural>,
<$X_{\eta+3}$(0,0), Real>,
<$X_{\eta+4}$(0,0), Natural>,
<$X_{\eta+5}$(0,0), Natural>,
<$X_{\eta+1}$(0,k-1), Binario>
}

P^y = {<$Y_{\eta+1}$(0,0), Binario>,
<$Y_{\eta+2}$(0,0), Binario >,
<$Y_{\eta+6}$(0,0), Real >,
<$Y_{\eta+7}$(0,0), Natural >,
<$Y_{\eta+8}$(0,0), Real >,
<$Y_{\eta+1}$(0,k-1), Natural>,
<$Y_{\eta+2}$(0,k-1), Binario>,
<$Y_{\eta+3}$(0,k-1), Natural>,
<$Y_{\eta+4}$(0,k-1), Natural>,
<$Y_{\eta+5}$(0,k-1), Natural>

}

Los ports se denominarán de la siguiente forma:

PORTS		
Port	**Nombre**	**Comentario**
Celda (0,0) : Primer celda del tramo		
$X_{\eta+1}$(0,0)	x-c-destino	Este port se utiliza para informar de la salida de un auto desde el cruce hacia el tramo, informando su destino (portvalue($X_{\eta+1}$) = Destino).
$X_{\eta+2}$ (0,0)	x-c-camino	Este port se utiliza para informar el camino del auto que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+2}$) = Camino).
$X_{\eta+3}$(0,0)	*x-c-velocidad*	*Este port se utiliza para informar la velocidad del vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+3}$) = Velocidad).*

$X_{\eta+4}(0,0)$	*x-c-tipo_vehiculo*	*Este port se utiliza para informar el tipo de vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+4}$) = Tipo Vehículo).*
$X_{\eta+5}(0,0)$	*x-c-tipo_conductor*	*Este port se utiliza para informar el tipo de conductor del vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+5}$) = Tipo de conductor).*
$Y_{\eta+1}$ (0,0)	y-c-haylugar	Este port se utiliza para que el cruce pueda saber si hay lugar en el tramo para que un auto pueda salir de él (portvalue($Y_{\eta+1}$) = 0).
$Y_{\eta+2}$ (0,0)	y-r-EntraAuto	Este port informa al controlador de congestion la entrada de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+6}$ *(0,0)*	*y-c-veloc_maxima*	*Este port informa al cruce (celda de salida) la velocidad máxima permitida en el tramo. Toma el valor del parámetro especificado en el tramo: max.*
$Y_{\eta+7}(0,0)$	*y-c-distancia*	*Le indica al cruce cual es la distancia del auto más cercano*
$Y_{\eta+8}(0,0)$	*y-c-veloc_auto_prox*	*Le indica al cruce cual es la velocidad del auto más cercano*
Celda (0,k-1) : Ultima celda del tramo		
$X_{\eta+1}$ (0,k-1)	x-c-haylugar	Este port permite saber si en el cruce hay lugar (portvalue($X_{\eta+1}$) = 0) para que avance un auto desde el tramo.
$Y_{\eta+1}$ (0,k-1)	y-c-destino	Este port informa al cruce de la presencia de un auto en el tramo que desea avanzar hacia él, informando su destino (portvalue($Y_{\eta+1}$) = Destino).
$Y_{\eta+2}$ (0,k-1)	y-r-SaleAuto	Este port informa al controlador de congestion la salida de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+3}$ (0,k-1)	y-c-camino	Este port se utiliza para informar el camino del auto que sale del tramo para ingresar al cruce (portvalue($Y_{\eta+3}$) = Camino).
$Y_{\eta+4}(0,k-1)$	*y-c-tipo_vehiculo*	*Este port se utiliza para informar el tipo de vehículo que sale hacia el cruce(portvalue($Y_{\eta+5}$) = Tipo Vehículo).*
$Y_{\eta+5}(0,k-1)$	*y-c-tipo_conductor*	*Este port se utiliza para informar el tipo de conductor del vehículo que sale hacia el cruce (portvalue($Y_{\eta+6}$) = Tipo de conductor).*

Nota: Los ports descriptos en letra *cursiva*, son los agregados por este nuevo modelo

$\boldsymbol{X} \in R$
$\boldsymbol{Y} \in R$
$\mathbf{n} = 1$
$\mathbf{t_1} = k$
$\boldsymbol{\eta} = v + 2$
N = { (0,-1), (0,0), (0,i) con $1 \leq i \leq v$ }
C = { C_{ij} / $i = 0 \wedge j \in [0, k-1]$ }, donde cada C_{ij} es un componente Cell-DEVS con demora. La especificación de cada celda ha sido descripta antes de introducir el modelo acoplado.
B = { (0,0), (0,k-1) }
Z se construye siguiendo la definición dada por el formalismo Cell_DEVS, instanciada con el vecindario de este espacio.
select = { (0,1), (0,0), (0,-1), (0, i) con $2 \leq i \leq v$ }

Celda (0,0)

El comportamiento para las celdas borde es distinto al definido anteriormente. Para la celda (0,0) se define el siguiente vecindario y comportamiento:

η = v+1
N = { (0,0), (0,i) 1≤i≤v }

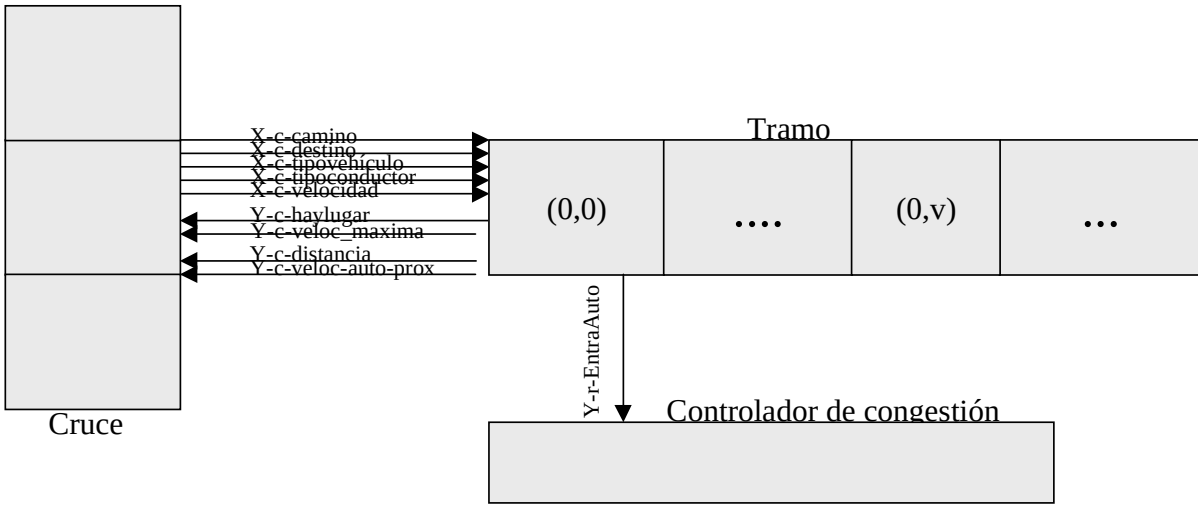


Figura 8- Vecindario y acoplamiento de la celda (0,0)

La función τ para esta celda se define como:

FUNCION τ			
Nuevo Estado	**Estado del vecindario**	**Nuevo estado de los ports**	**Nombre de la regla**
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_Vehículo* = 0 *Tipo_Conductor* = 0	(Destino(0,0) != 0 and Destino(0,1) = 0)	/* Info al cruce */ Sea d = Distancia ({ Destino(0, i), 1≤i≤v }) que indica en que celda se encuentra el auto más cercano que tiene delante. Send(y-c-distancia, d) Send(y-c-veloc_auto_prox, Velocidad(0,d))	Sale_Hacia_Adelante
Sea d = Distancia ({ Destino(0, i), 1≤i≤v }) que indica en que celda se encuentra el auto más cercano que tiene delante. El nuevo estado es: *Destino* = portvalue(x-c-destino) *Camino* = portvalue(x-c-camino) *Velocidad* = Velocidad (portvalue(x-g-velocidad), Portvalue(x-c-tipo_Vehiculo), Portvalue(x-c-tipo_Conductor), d, Velocidad(0,d),	(portvalue(x-c-destino) != 0 and Destino(0,0) = 0)	/* Info de congestion */ Send(1, y-r-EntraAuto) /* Info al cruce */ Send(y-c-distancia, 1) Send(y-c-veloc_auto_prox, Velocidad(0,0))	Llega_Desde_Cruce

Max) *Tipo_Vehículo* = portvalue(x-c-tipo_Vehiculo) *Tipo_Conductor* = portvalue(x-c-tipo_Conductor)			
(0,0)	t /*en otro caso conserva estado*/	/* Info al cruce */ Sea d = Distancia ({ Destino(0, i), 0≤i≤v }) que indica en que celda se encuentra el auto más cercano que tiene delante. Send(y-c-distancia, d) Send(y-c-veloc_auto_prox, Velocidad(0,d))	Default

Los demás parámetros del modelo para esta celda no cambian.

La celda (0,0) tiene el vecindario formado por ella y las celdas de adelante de acuerdo a su visibilidad, además se encuentra acoplada a la celda del cruce desde la que recibe los vehículos y al controlador de congestión. Esto se puede ver en la figura 8. Para conocer el estado de la celda del cruce se utiliza el port x-c-destino, siempre que sea distinto de 0 representa que existe un vehículo que saldrá del cruce hacia la celda (0,0), y por consiguiente se informará todas las características del auto contenido en ella. Tambien se informará al controlador de congestion de la entrada de un auto al tramo. Por lo tanto, la regla Llega_Desde_Cruce representa el movimiento de avance de un vehículo que se encuentra en el cruce hacia la celda (0,0). Para calcular la nueva velocidad se utiliza la función 'Distancia' que devuelve la posición del auto más cercano y a través del port de esa celda se conoce su velocidad.
En todas las reglas, se envía al cruce información acerca de la distancia a la que se encuentra el auto más cercano del tramo.

La celda (0, k-1) posee un vecindario y comportamiento similar al definido en el caso de la propuesta 1 (Modelo GAP). La única diferencia con este caso, es que en este modelo no existe el modelo 'GAP', por lo tanto, no se envía la información especificada para los ports que conectar a ambos modelos.

Los modelos correspondientes a los tramos de 2, 3, 4 y más de 4 carriles sufren modificaciones análogas a las del caso de un carril:

- Cambia la definición del estado.
- Se extiende el vecindario de las celdas.

La definición de los modelos correspondientes a estos casos no será presentada en esta tesis y queda como propuesta para trabajos futuros.

Definición del modelo que representa a los cruces

La definición del modelo correspondiente a los cruces es exactamente la misma que la presentada en la propuesta 1 (Modelo GAP).
La única diferencia radica en su acoplamiento con otros modelos:

- En la propuesta 1 recibe información acerca de la posición del auto más cercano en el tramo de salida a través del modelo GAP, junto con la velocidad que lleva el mismo. En este caso esta información se la enviará directamente el tramo.

Por lo tanto, se reemplaza la denominación de los siguientes ports:

Port	Nombre en propuesta 1	Nuevo Nombre	Comentario
$X_{\eta+7}(0,i)$, $i \in$ Out	x-g-distancia	x-t-distancia	Este port se utiliza para conocer la distancia a la que se encuentra el auto más cercano (de adelante) en el tramo de salida.
$X_{\eta+8}(0,i)$, $i \in$ Out	x-g-velocidad	x-t-veloc_auto_prox	Este port se utiliza para conocer la velocidad que tiene el auto más cercano (de adelante) que se encuentra en el tramo de salida

Y se redefine el acoplamiento entre los tramos y los cruces:

Acoplamiento de los tramos con los cruces

Notas:
- Sólo se especificarán los nuevos acoplamientos, con respecto al modelo presentado en la sección de Ruteo dinámico.
- La modificación del acoplamiento introducido por esta propuesta, se detalló en letra *cursiva*.

Ports de salida tramo – ingreso cruce
- $Y_{\eta+4}$ (0,k-1) (y-c-tipo_vehiculo) → $X_{\eta+4}$ (Celda de entrada: x-t-tipo_vehiculo) Informando el tipo del vehículo que sale desde el tramo hacia el cruce.
- $Y_{\eta+5}$ (0,k-1) (y-c-tipo_conductor) → $X_{\eta+5}$ (Celda de entrada: x-t-tipo_conductor) Informando el tipo de conductor del auto que sale desde el tramo hacia el cruce.
- $Y_{\eta+6}$ (0,0) (y-c-veloc_maxima) → $X_{\eta+9}$ (Celda de salida: x-t-veloc_maxima) Informando la velocidad máxima permitida en el tramo hacia el cual va a salir el auto desde el cruce.
- *$Y_{\eta+7}$ (0,0) (y-c-distancia) → $X_{\eta+7}$ (Cruce) (Celda de salida: x-t-distancia), informando la distancia del auto más cercano a la celda dentro del tramo de salida del cruce.*
- *$Y_{\eta+8}$ (0,0) (y-c-veloc_auto_prox) → $X_{\eta+8}$ (Cruce) (Celda de salida: x-t-veloc_auto_prox), informando la velocidad del auto más cercano a la celda dentro del tramo de salida del cruce.*

Ports de salida cruce – ingreso tramo
- $Y_{\eta+6}$ (Celda de salida: y-t-velocidad) → $X_{\eta+3}$ (0,0) (x-c-velocidad) Informando la velocidad del auto que ingresa desde el cruce hacia el tramo.
- $Y_{\eta+4}$ (Celda de salida: y-t-tipo_vehículo) → $X_{\eta+4}$ (0,0) (x-c-tipo_vehiculo) Informando el tipo de vehículo que ingresa desde el cruce hacia el tramo.
- $Y_{\eta+5}$ (Celda de salida: y-t-tipo_conductor) → $X_{\eta+5}$ (0,0) (x-c-tipo_conductor) Informando el tipo de conductor del auto que ingresa desde el cruce hacia el tramo.

Propuesta 3 - Propagación de información

Este modelo ha sido definido tomando como base, el presentado en la sección 4.2 (ruteo dinámico). Se agrega como característica de los vehículos:

- El tipo de conductor que lo conduce, y
- El tipo de vehículo (por ejemplo, auto, camioneta, colectivo, etc)

Por otro lado, se agrega información que permite conocer a qué distancia se encuentra el auto más cercano (hacia adelante) y que velocidad lleva. Para poder obtener esta información, se agrega al estado de las celdas de los tramos la distancia a la que se encuentra el auto más cercano y su respectiva velocidad. Si la celda vecina (hacia adelante) tiene un auto, la distancia es 1, sino se suma 1 a la distancia del auto más cercano respecto a la celda vecina. De esta forma, se va propagando la información a través de las celdas vecinas (aunque el origen de la misma se encuentre fuera del vecindario.

Los modelos de las calles y los cruces debieron ser modificados de la siguiente forma:

Calles de carril único

Un tramo ***t = (p1, p2, 1, a, dir, max, nro_tramo)*** de carril único se define como un modelo ***Cell_DEVS*** de una dimensión con demora de transporte, cuya estructura se presenta en la siguiente figura 1.

Cada celda en este espacio se define como:

$$C_{0j}\ (nro\_tramo) = < I, X, S, Y, N, \delta_{in}, \delta_{ex}, delay, d, \tau, \lambda, D >$$

con

I = <η, P^x, P^y>, donde

η = 3; y P^x y P^y donde

P^x = { (X_1, Record), (X_2, Record), (X_3, Record) }

P^y = { (Y_1, Record), (Y_2, Record), (Y_3, Record) }.

X, Y ∈ N

S = (destino, camino, velocidad, tipo_vehiculo, tipo_conductor, dist_auto_cercano, veloc_auto_cercano)

Donde

destino = { != 0 si hay un vehículo en la celda ; 0 sino.

Destino ∈ N: Destino del auto en la celda (cruce)

| {$t_1.t_2 \ldots t_n$}donde ti ∈ N ∧ (∀i (∃ r ∈ Tramos/ Nro_tramo(r) = t_i))

Camino = { si hay un vehículo en la celda
0 sino.

representa el camino que sigue el auto contenido en la celda

velocidad ∈ R: velocidad del auto que se encuentra en la celda

tipo_vehiculo ∈ N: tipo de vehículo

tipo_conductor ∈ N: tipo de conductor

dist_auto_cercano ∈ N: distancia (en celdas) a la que se encuentra el auto más cercano que se tiene hacia adelante.

veloc_auto_cercano ∈ R: velocidad que posee el auto más cercano que se tiene hacia adelante.

Notas:

- destino = 0 ⇒ Camino = 0 (si no hay auto en la celda tampoco hay camino)
- destino = 0 ⇒ Velocidad = 0 (si no hay auto en la celda, la velocidad es cero)
- destino = 0 ⇒ tipo_vehiculo = 0 (si no hay auto en la celda, el tipo de vehículo es cero)
- destino = 0 ⇒ tipo_conductor = 0 (si no hay auto en la celda, el tipo de conductor es cero)
- dist_auto_cercano = 0 ⇒ No hay ningún auto delante de él, dentro del tramo.

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

(0,-1)	(0,0)	(0,1)

Figura 9- Vecindario de la celda origen

delay = transport

d = ***Conversión_Demora (Velocidad (velocidad_actual,***
tipo_vehiculo,
tipo_conductor,
distancia_auto_cercano,
velocidad_auto_cercano,
Velocidad_maxima_tramo))

donde,

- ***Velocidad:*** es una función que calcula la nueva velocidad del auto en base a la velocidad actual del mismo, al tipo de vehículo, el tipo de conductor, la distancia y velocidad del auto más cercano hacia adelante y la velocidad máxima permitida en el mismo.
- ***Conversión_Demora:*** es una función detallada en el Apéndice E, que recibe como parámetro un valor natural que representa una velocidad (en km/h) y devuelve el

tiempo (en segundos) que se debe demorar el auto en la celda para representar que avanza con esa velocidad.

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

τ: S x N $\rightarrow$ S se define de la siguiente manera,

Nuevo estado	Estado del vecindario	Nombre de la regla
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Velocidad* = Velocidad (Velocidad(0,-1), Tipo_Vehiculo(0,-1), Tipo_Conductor(0,-1), 1, Velocidad(0,1), Max) *Tipo_Vehículo* = Tipo_Vehiculo(0,-1) *Tipo_Conductor* = Tipo_Conductor(0,-1) *Dist_auto_cercano* = 1 *Veloc_auto_cercano* = Velocidad (0,1)	Destino (0,-1) != 0 and Destino(0,0) = 0 and Destino(0,1) = 1	Llega_Desde_Atrás_con_auto_adelante
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Velocidad* = Velocidad (Velocidad(0,-1), Tipo_Vehiculo(0,-1), Tipo_Conductor(0,-1), IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Dist_Auto_Cercano(0,1) +1), IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Veloc_Auto_Cercano(0,1)), Max) *Tipo_Vehículo* = Tipo_Vehiculo(0,-1) *Tipo_Conductor* = Tipo_Conductor(0,-1) *Dist_auto_cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Dist_Auto_Cercano(0,1) +1), *Veloc_auto_cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Veloc_Auto_Cercano(0,1))	Destino (0,-1) != 0 and Destino(0,0) = 0 and Destino(0,1) = 0	Llega_Desde_Atrás_sin_auto_adelante

Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_Vehículo* = 0 *Tipo_Conductor* = 0 *Dist_auto_cercano* = 1 *Veloc_auto_cercano* = Velocidad (Velocidad(0,0), Tipo_Vehiculo(0,0), Tipo_Conductor(0,0), Dist_Auto_Cercano(0,1), Veloc_Auto_Cercano(0,1), Max)	Destino(0,0) != 0 and Destino(0,1) = 0	Sale_Hacia_Adelante
Destino = Destino(0,0) *Camino* = Camino(0,0) *Velocidad* = Velocidad (0,0) *Tipo_Vehículo* = Tipo_Vehiculo(0,0) *Tipo_Conductor* = Tipo_Conductor(0,0) *Dist_auto_cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Dist_Auto_Cercano(0,1) +1) *Veloc_auto_cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Veloc_Auto_Cercano(0,1))	t and Destino(0,1) = 0 /*en otro caso conserva estado*/	Default_sin_auto_delante
Destino = Destino(0,0) *Camino* = Camino(0,0) *Velocidad* = Velocidad (0,0) *Tipo_Vehículo* = Tipo_Vehiculo(0,0) *Tipo_Conductor* = Tipo_Conductor(0,0) *Dist_auto_cercano* = 1 *Veloc_auto_cercano* = Velocidad (0,1)	t and Destino(0,1) != 0 /*en otro caso conserva estado*/	Default_con_auto_delante

Para actualizar la información sobre el auto de adelante más cercano, se dividen dos casos:

1. La celda de adelante tiene un auto
2. La celda de adelante está vacía

En el primer caso se pone como distancia del auto cercano 1 y como velocidad la velocidad del auto de la celda de adelante.
En el segundo caso a la distancia del auto más cercano que tiene la celda inmediatamente anterior se le suma 1 y se copia la velocidad del auto más cercano (sólo en el caso que la distancia sea distinta de 0, ya que eso significa que no existe un auto delante dentro del tramo).
Aún cuando no cambie la información sobre el auto contenido en la celda, se actualizan los datos sobre el auto más cercano que se encuentra a la misma.
Estos datos (Dist_Auto_Cercano y Veloc_Auto_Cercano), incluidos en el estado, se utilizan para calcular la velocidad que tomará en auto de la misma manera descripta en las propuestas anteriores.
La función 'Velocidad' ya fue presentada en las propuestas anteriores.

La función IF_ELSE se define de la siguiente forma:

IF_ELSE(expresión, condición, valor1, valor2)
IF *expresión = condición* THEN
  RETURN *valor1*
ELSE
  RETURN *valor2*

END IF

Modelo Acoplado

El modelo acoplado se define de la misma forma que para la propuesta 2 (Extensión del vecindario), modificando solamente las reglas correspondientes a la celda (0,0) del tramo.
Se presenta solamente la modificación requerida con respecto a la propuesta anterior.

Celda (0,0)

La función τ para la celda (0,0) se define como:

FUNCION τ			
Nuevo Estado	**Estado del vecindario**	**Nuevo estado de los ports**	**Nombre de la regla**
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_Vehículo* = 0 *Tipo_Conductor* = 0 *Dist_Auto_cercano* = 1 *Veloc_Auto_Cercano* = Velocidad (Velocidad(0,0), Tipo_Vehiculo(0,0), Tipo_Conducto(0,0), Dist_Auto_Cercano(0,1), Veloc_Auto_Cercano(0,1), Max)	(Destino(0,0) != 0 and Destino(0,1) = 0)	/* Info al cruce */ Send(y-c-distancia, 2) Send(y-c-veloc_auto_prox, Veloc_Auto_Cercano(0,0)) Nota: Se asume que los valores de la celda (0,0) son los que tiene luego de ejecutada la regla.	Sale_Hacia_Adelante
Destino = portvalue(x-c-destino) *Camino* = portvalue(x-c-camino) *Velocidad* = Velocidad (portvalue(x-g-velocidad), Portvalue(x-c-tipo_Vehiculo), Portvalue(x-c-tipo_Conductor), 1, Velocidad(0,1), Max) *Tipo_Vehículo* = portvalue(x-c-tipo_Vehiculo) *Tipo_Conductor* = portvalue(x-c-tipo_Conductor) *Dist_Auto_Cercano* = 1 *Veloc_Auto_Cercano* = Velocidad(0,1)	(portvalue(x-c-destino) != 0 and Destino(0,0) = 0 and Destino(0,1) != 0)	/* Info de congestion */ Send(1, y-r-EntraAuto) /* Info al cruce */ Send(y-c-distancia, 1) Send(y-c-veloc_auto_prox, Velocidad(0,0)) Nota: Velocidad(0,0) es la nueva velocidad calculada.	Llega_Desde_Cruce_con_auto_delante
Destino = portvalue(x-c-destino) *Camino* = portvalue(x-c-camino) *Velocidad* = Velocidad (portvalue(x-g-velocidad), Portvalue(x-c-tipo_Vehiculo), Portvalue(x-c-tipo_Conductor), IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Dist_Auto_Cercano(0,1) +1) ,	(portvalue(x-c-destino) != 0 and Destino(0,0) = 0 and Destino(0,1) = 0)	/* Info de congestion */ Send(1, y-r-EntraAuto) /* Info al cruce */ Send(y-c-distancia, 1) Send(y-c-veloc_auto_prox, Velocidad(0,0)) Nota: Velocidad(0,0) es la nueva velocidad calculada.	Llega_Desde_Cruce_sin_auto_delante

IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Veloc_Auto_Cercano(0,1)) , Max) *Tipo_Vehículo* = portvalue(x-c-tipo_Vehiculo) *Tipo_Conductor* = portvalue(x-c-tipo_Conductor) *Dist_Auto_Cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Dist_Auto_Cercano(0,1) +1) *Veloc_Auto_Cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Veloc_Auto_Cercano(0,1))			
Destino = Destino(0,0) *Camino* = Camino (0,0) *Velocidad* = Velocidad (0,0) *Tipo_Vehículo* = Tipo_Vehiculo(0,0) *Tipo_Conductor* = Tipo_Conductor(0,0) *Dist_Auto_Cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Dist_Auto_Cercano(0,1) +1) *Veloc_Auto_Cercano* = IF_ELSE(Dist_Auto_cercano(0,1), 0, 0, Veloc_Auto_Cercano(0,1))	t and Destino(0,1) = 0 /*en otro caso conserva estado*/	/* Info al cruce */ Send(y-c-distancia, IF_ELSE(Destino(0,0), 0, Dist_Auto_Cercano(0,0)+1, 1)) Send(y-c-veloc_auto_prox, IF_ELSE(Destino(0,0), 0, Veloc_Auto_Cercano(0,0), Velocidad(0,0))) Nota: Se asume que los valores de la celda (0,0) son los que tiene luego de ejecutada la regla.	Default_sin_auto_delante
Destino = Destino(0,0) *Camino* = Camino (0,0) *Velocidad* = Velocidad (0,0) *Tipo_Vehículo* = Tipo_Vehiculo(0,0) *Tipo_Conductor* = Tipo_Conductor(0,0) *Dist_Auto_Cercano* = 1 *Veloc_Auto_Cercano* = Velocidad(0,1)	t and Destino(0,1) != 0 /*en otro caso conserva estado*/	/* Info al cruce */ Send(y-c-distancia, IF_ELSE(Destino(0,0), 0, Dist_Auto_Cercano(0,0)+1, 1)) Send(y-c-veloc_auto_prox, IF_ELSE(Destino(0,0), 0, Veloc_Auto_Cercano(0,0), Velocidad(0,0))) Nota: Se asume que los valores de la celda (0,0) son los que tiene luego de ejecutada la regla.	Default_con_auto_delante

La modificación que presentan estas reglas constituye el envío de información al cruce.

- *Sale hacia adelante:* Se envía como distancia el valor 2 (ya que la celda (0,0) queda vacía y la (0,1) se ocupa), y como velocidad la calculada en base a los datos del vehículo (con esa velocidad el vehículo ingresa a la celda (0,1)) .
- *Llega_Desde_Cruce_con_auto_delante y Llega_Desde_Cruce_sin_auto_delante:* En ambos casos, se ocupa la celda (0,0), por lo tanto, la distancia desde el cruce es 1 y la velocidad es la que toma el vehículo dentro de la celda.

- *Default_con_auto_delante y Default_con_auto_delante:* Si la celda (0,0) está vacía, le suma 1 a la distancia del auto más cercano, toma la velocidad del mismo y envía esta información al cruce. Si la celda (0,0) está ocupada, envía como distancia 1 y como la velocidad, la velocidad que posee el auto dentro de la celda.

La modificación de las condiciones de las reglas y los nuevos estados no varían con respecto a las presentadas para las celdas intermedias del tramo.

Celda (0,k-1)

La especificación de la celda (0,k-1) no varía a la presentada en las propuestas anteriores. Esto se debe a que, por ser la última celda del tramo no necesita la información acerca del auto más cercano (hacia adelante) ni su velocidad.

Los modelos correspondientes a los tramos de 2, 3, 4 y más de 4 carriles sufren modificaciones análogas a las del caso de un carril:

- Cambia la definición del estado.
- Se propaga la información dentro del carril. Dependiendo de la definición del vecindario en las celdas laterales, se tendrá información acerca de los demás carriles.

La definición de los modelos correspondientes a estos casos no será presentada en esta tesis y queda como propuesta para trabajos futuros.

Cruces y acoplamientos entre Tramos y Cruces

Tanto la definición del modelo que representa a los cruces como el acoplamiento entre cruces y tramos, no varían con respecto a la propuesta 2 (Extensión del vecindario).
Esto es, porque la única diferencia entre las dos propuesta consiste en la forma que obtienen información acerca del auto que se tiene más cercano (hacia adelante) dentro del tramo. Por lo tanto, sólo se modifica la definición del modelo que representa a los tramos, conservando la misma interface (ports de entrada y salida) en ambas propuestas.

Propuesta 4 - Estimación con información de congestión

Este modelo ha sido definido tomando como base, el presentado en la sección 4.2 (ruteo dinámico). Se agrega como característica de los vehículos:

- El tipo de conductor que lo conduce, y
- El tipo de vehículo (por ejemplo, auto, camioneta, colectivo, etc)

A diferencia de las propuestas anteriores, en este caso, no se conoce a que distancia se encuentra el auto más cercano. La función que determina la velocidad que tomará el auto, usará como parámetro la información de congestión, brindada por el controlador de congestión como una estimación del estado en el que se encuentra el tramo.
Como la información usada en esta propuesta es menos precisa que en los casos anteriores, se agregó un factor correctivo que ayude a mejorar la estimación.
El objetivo de este factor, es representar la cantidad de veces que se logra avanzar con éxito (cada vez que ocurre esta acción le suma 1 al factor) y la cantidad que se debe esperar por estar la celda anterior ocupada (cada vez que ocurre esta acción le resta 1 al factor). De esta manera, un factor alto, indica que el camino está despejado, por lo tanto, la velocidad puede ser alta. Mientras que un factor pequeño está indicando que el tránsito lo está demorando, por lo tanto, debe circular a una velocidad baja.

Los modelos de las calles debieron ser modificados de la siguiente forma:

Calles de carril único

Un tramo ***t = (p1, p2, 1, a, dir, max, nro_tramo)*** de carril único se define como un modelo ***Cell_DEVS*** de una dimensión con demora de transporte, cuya estructura se presenta en la siguiente figura 1.

Cada celda en este espacio se define como:

$$C_{0j}\ (nro\_tramo) = < I, X, S, Y, N, \delta_{int}, \delta_{ext}, delay, d, \tau, \lambda, D >$$

con

I = $<\eta, P^x, P^y>$, donde

$\eta = 3$; y P^x y P^y donde
$P^x = \{ (X_1, Record), (X_2, Record), (X_3, Record) \}$
$P^y = \{ (Y_1, Record), (Y_2, Record), (Y_3, Record) \}$

X, Y $\in$ N

S = (destino, camino, velocidad, tipo_vehiculo, tipo_conductor, factor_aceleracion)

Donde

destino = { != 0 si hay un vehículo en la celda; / 0 sino.

Destino ∈ N: Destino del auto en la celda (cruce)

Camino = $\begin{cases} \{t_1.t_2 \ldots t_n\} \text{donde } ti \in N \wedge (\forall i\ (\exists\ r \in \text{Tramos/ Nro\_tramo}(r) = t_i)) & \text{si hay un vehículo en la celda} \\ 0 \text{ sino.} \end{cases}$

velocidad ∈ R: velocidad del auto que se encuentra en la celda

tipo_vehiculo ∈ N: tipo de vehículo

tipo_conductor ∈ N: tipo de conductor

factor_aceleracion ∈ N: representa la cantidad de intentos de avance con éxito menos los fallidos.

Notas:

- destino = 0 ⇒ Camino = 0 (si no hay auto en la celda tampoco hay camino)
- destino = 0 ⇒ Velocidad = 0 (si no hay auto en la celda, la velocidad es cero)
- destino = 0 ⇒ tipo_vehiculo = 0 (si no hay auto en la celda, el tipo de vehículo es cero)
- destino = 0 ⇒ tipo_conductor = 0 (si no hay auto en la celda, el tipo de conductor es cero)
- destino = 0 ⇒ factor_aceleracion = 0

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

(0,-1)	(0,0)	(0,1)

Figura 10 - Vecindario de la celda origen

delay = transport

d = ***Conversión_Demora (Velocidad (velocidad_actual,***
n,
k,
peso,
tipo_vehiculo,
tipo_conductor,
factor_aceleracion,
Velocidad_maxima_tramo))

donde,

- ***Velocidad:*** es una función que calcula la nueva velocidad del auto en base a la velocidad actual del mismo, a la cantidad de celdas del tramo (k), el número de carriles del mismo (n), la congestión que presenta (dado por el parámetro peso), el tipo de vehículo, el tipo de conductor, el factor correctivo anteriormente explicado y la velocidad máxima permitida en el tramo.
- ***Conversión_Demora:*** es una función detallada en el Apéndice E, que recibe como parámetro un valor natural que representa una velocidad (en km/h) y devuelve el

tiempo (en segundos) que se debe demorar el auto en la celda para representar que avanza con esa velocidad.

λ, δ_{int} y δ_{ext} se comportan como las funciones definidas por el formalismo Cell-DEVS para demoras de transporte.

τ: S x N $\rightarrow$ S se definirá junto con el modelo acoplado, por utilizar los ports conectados al controlador de congestión.

Modelo acoplado

El modelo acoplado correspondiente al tramo *t = (p1, p2, 1, a, dir, max, nro_tramo)* se define como:

$$TC1(k, max, nro\_tramo) = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, \eta, N, C, B, Z, select >$$

donde,

Ylist = { (0,0), (0,k-1) }

Xlist = { (0,0), (0,k-1) }

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

P^x = {$<X_{\eta+1}(0,0)$, Natural>,
$<X_{\eta+2}(0,0)$, Natural>,
$<X_{\eta+3}(0,0)$, Real>,
$<X_{\eta+4}(0,0)$, Natural>,
$<X_{\eta+5}(0,0)$, Natural>,
$<X_{\eta+6}(0,i)$, Natural> con $0 \leq i < k-1$,
$<X_{\eta+1}(0,k-1)$, Binario>
}

P^y = {$<Y_{\eta+1}(0,0)$, Binario>,
$<Y_{\eta+2}(0,0)$, Binario >,
$<Y_{\eta+6}(0,0)$, Real >,
$<Y_{\eta+1}(0,k-1)$, Natural>,
$<Y_{\eta+2}(0,k-1)$, Binario>,
$<Y_{\eta+3}(0,k-1)$, Natural>,
$<Y_{\eta+4}(0,k-1)$, Natural>,
$<Y_{\eta+5}(0,k-1)$, Natural>

}

Los ports se denominarán de la siguiente forma:

PORTS		
Port	**Nombre**	**Comentario**

Celda (0,0) : Primer celda del tramo		
$X_{\eta+1}(0,0)$	x-c-destino	Este port se utiliza para informar de la salida de un auto desde el cruce hacia el tramo, informando su destino (portvalue($X_{\eta+1}$) = Destino).
$X_{\eta+2}$ (0,0)	x-c-camino	Este port se utiliza para informar el camino del auto que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+2}$) = Camino).
$X_{\eta+3}(0,0)$	*x-c-velocidad*	*Este port se utiliza para informar la velocidad del vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+3}$) = Velocidad).*
$X_{\eta+4}(0,0)$	*x-c-tipo_vehiculo*	*Este port se utiliza para informar el tipo de vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+4}$) = Tipo Vehículo).*
$X_{\eta+5}(0,0)$	*x-c-tipo_conductor*	*Este port se utiliza para informar el tipo de conductor del vehículo que ingresa desde el cruce hacia el tramo (portvalue($X_{\eta+5}$) = Tipo de conductor).*
$Y_{\eta+1}$ (0,0)	y-c-haylugar	Este port se utiliza para que el cruce pueda saber si hay lugar en el tramo para que un auto pueda salir de él (portvalue($Y_{\eta+1}$) = 0).
$Y_{\eta+2}$ (0,0)	y-r-EntraAuto	Este port informa al controlador de congestion la entrada de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+6}$ (0,0)	*y-c-veloc_maxima*	*Este port informa al cruce (celda de salida) la velocidad máxima permitida en el tramo. Toma el valor del parámetro especificado en el tramo: max.*
Celdas (0,i) con $0 \leq i < k-1$		
$X_{\eta+5}(0,i)$	*x-r-peso*	*Este port se utiliza para que la celda conozca la información de congestión del tramo*
Celda (0,k-1) : Ultima celda del tramo		
$X_{\eta+1}$ (0,k-1)	x-c-haylugar	Este port permite saber si en el cruce hay lugar (portvalue($X_{\eta+1}$) = 0) para que avance un auto desde el tramo.
$Y_{\eta+1}$ (0,k-1)	y-c-destino	Este port informa al cruce de la presencia de un auto en el tramo que desea avanzar hacia él, informando su destino (portvalue($Y_{\eta+1}$) = Destino).
$Y_{\eta+2}$ (0,k-1)	y-r-SaleAuto	Este port informa al controlador de congestion la salida de un auto al tramo (portvalue($Y_{\eta+2}$) = 1).
$Y_{\eta+3}$ (0,k-1)	y-c-camino	Este port se utiliza para informar el camino del auto que sale del tramo para ingresar al cruce (portvalue($Y_{\eta+3}$) = Camino).
$Y_{\eta+4}(0,k-1)$	*y-c-tipo_vehiculo*	*Este port se utiliza para informar el tipo de vehículo que sale hacia el cruce(portvalue($Y_{\eta+5}$) = Tipo Vehículo).*
$Y_{\eta+5}(0,k-1)$	*y-c-tipo_conductor*	*Este port se utiliza para informar el tipo de conductor del vehículo que sale hacia el cruce (portvalue($Y_{\eta+6}$) = Tipo de conductor).*

Nota: Los ports descriptos en letra *cursiva*, son los agregados por este nuevo modelo

$\boldsymbol{X} \in R$
$\boldsymbol{Y} \in R$
$\mathbf{n} = 1$
$\mathbf{t_1} = k$
$\boldsymbol{\eta} = 3$
$\mathbf{N} = \{ (0,-1), (0,0), (0,1) \}$
$\mathbf{C} = \{ C_{ij} / i = 0 \wedge j \in [0, k-1] \}$, donde cada C_{ij} es un componente Cell-DEVS con demora.
$\mathbf{B} = \{ (0,0), (0,k-1) \}$
Z se construye siguiendo la definición dada por el formalismo Cell_DEVS, instanciada con el vecindario de este espacio.

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

Se define el vecindario y comportamiento de las celdas de la siguiente manera:

Celda (0,0)

η = 2

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

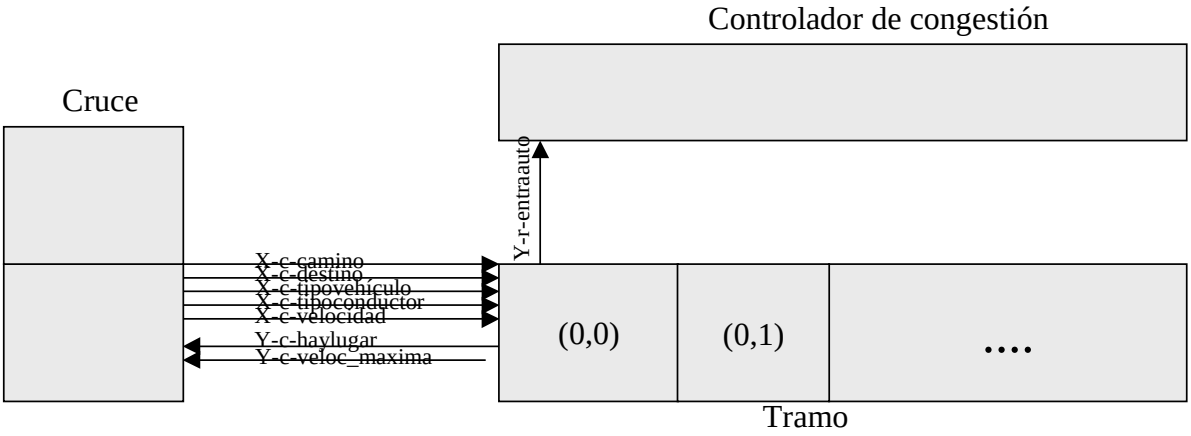


Figura 11- Vecindario y acoplamiento de la celda (0,0)

FUNCION τ			
Nuevo Estado	**Estado del vecindario**	**Nuevo estado de los ports**	**Nombre de la regla**
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_vehiculo* = 0 *Tipo_conductor* = 0 *Factor_aceleracion* = 0	Destino(0,0) != 0 and Destino(0,1) = 0		Sale_Hacia_ Adelante
Destino = portvalue(x-c-destino) *Camino* = portvalue(x-c-camino) *Velocidad* = *Velocidad* (portvalue(x-c-velocidad), n, k, portvalue(x-r-peso), portvalue(x-c-tipo_vehiculo), portvalue(x-c-tipo_conductor), FACTOR_ ACELERACION_ INICIO_TRAMO, Max)	portvalue(x-c-destino) != 0 and Destino(0,0) = 0	Send(1, y-r-EntraAuto)	Llega_ Desde_ Cruce

Tipo_vehiculo = portvalue(x-c-tipo_vehiculo) *Tipo_conductor* = portvalue(x-c-tipo_conductor) *Factor_aceleracion* = FACTOR_ ACELERACION_ INICIO_TRAMO			
Destino = Destino(0,0) *Camino* = Camino(0,0) *Velocidad* = Velocidad (0,0) *Tipo_vehiculo* = tipo_vehiculo (0,0) *Tipo_conductor* = tipo_conductor (0,0) *Factor_aceleracion* = factor_aceleracion (0,0) - 1	Destino(0,0) = 1 And Destino(0,1) = 1		Mantiene_ Posicion
(0,0)	t /*en otro caso conserva estado*/		Default

La regla Llega_Desde_Cruce representa el movimiento de avance de un vehículo que se encuentra en el cruce hacia la celda (0,0). Al ingresar un auto a un tramo, el campo factor_aceleración que forman parte del estado de la celda, se inicializa con un valor que es parámetro del sistema: FACTOR_ACELERACION_INICIO_TRAMO.
La regla Mantiene_Posicion representa el caso en que existe una auto en la celda (0,1) y esto impide el movimiento del auto que se encuentra en la celda (0,0). En este caso, del estado de la celda, solo se altera el factor de aceleración. El resto del estado de la celda permanece intacto.

Celdas (0,i) con 1 ≤ i ≤ k-2 (Celdas intermedias)

η = 3

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

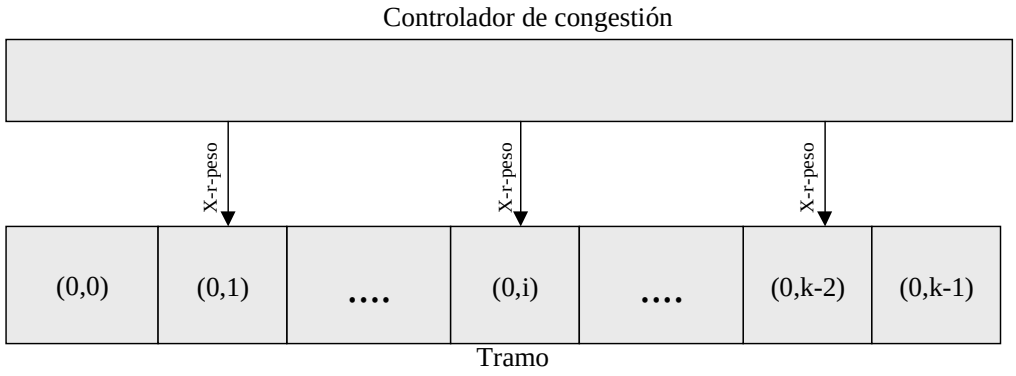


Figura 12- Vecindario y acoplamiento de las celdas intermedias

FUNCION τ		
Nuevo estado	**Estado del vecindario**	**Nombre de la regla**
Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Velocidad* = *Velocidad* (velocidad(0,-1), n, k, portvalue(x-r-peso), tipo_vehiculo (0,-1), tipo_conductor (0,-1), factor_aceleracion (0,-1), Max) *Tipo_vehiculo* = tipo_vehiculo (0,-1) *Tipo_conductor* = tipo_conductor (0,-1) *Factor_aceleracion* = factor_aceleracion (0,-1) + 1	Destino (0,-1) != 0 and Destino(0,0) = 0	**Llega_desde_atras**
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_vehiculo* = 0 *Tipo_conductor* = 0 *Factor_aceleracion* = 0	Destino(0,0) != 0 and Destino(0,1) = 0	**Sale_Hacia_adelante**
Destino = Destino(0,0) *Camino* = Camino(0,0) *Velocidad* = Velocidad (0,0) *Tipo_vehiculo* = tipo_vehiculo (0,0) *Tipo_conductor* = tipo_conductor (0,0) *Factor_aceleracion* = factor_aceleracion (0,0) - 1	Destino(0,0) != 0 and Destino(0,1) = 1	Mantiene_Posicion
(0,0)	T /*en otro caso conserva estado*/	Default

La nueva velocidad del auto se obtiene utilizando la funcion *Velocidad*, que calcula la nueva velocidad en base a la siguiente información:

- La velocidad actual del vehículo
- El numero de carriles del tramo (n)
- La cantidad de celdas del tramo (k)
- La congestión que presenta el tramo (dado por el parámetro peso)
- El tipo de vehículo
- El tipo de conductor
- El factor de corrección ya explicado.
- La velocidad máxima de circulación permitida dentro del tramo

El movimiento de los vehículos (especificado con la función τ) queda definido por 4 reglas:

- La regla *Llega_Desde_Atrás* representa el movimiento recto del vehículo desde la celda de atrás hacia la origen, siempre y cuando ésta se encuentre vacía. Se actualiza el factor de aceleración, sumándole 1 ya que el vehículo logró avanzar con éxito.
- La regla *Sale_Hacia_Adelante* representa el movimiento recto del vehículo que se encuentra en la celda origen hacia la de adelante.
- La regla *Mantiene_Posicion* representa el caso en que el auto no puede avanzar debido a la presencia de un auto en la celda de adelante. Como esta regla representa el intento fallido de avance, se actualiza el factor de corrección, restándole una unidad.

- La regla Default considera cualquier otro caso donde el estado de la celda no cambia.

Celda (0,k-1)

η = 2

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

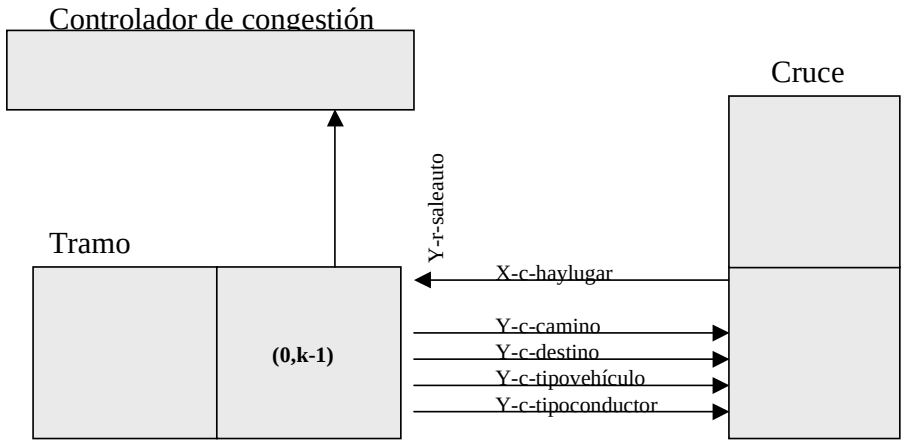


Figura 13- Vecindario y acoplamiento de la celda (0,k-1)

FUNCION τ y tipo de demora (delay)				
Nuevo Estado	**Estado del vecindario**	**Nuevo Estado de los Ports**	**Delay**	**Nombre de la regla**
Destino = 0 *Camino* = 0 *Velocidad* = 0 *Tipo_vehiculo* = 0 *Tipo_conductor* = 0 *Factor_aceleracion* = 0	Destino(0,0) != 0 And portvalue(x-c-haylugar) = 0	/* Info al cruce */ Send (Destino(0,0), y-c-destino) Send (Camino(0,0), y-c-camino)) Send (Tipo_Vehiculo(0,0), y-c-tipo_vehículo)) Send (Tipo_Conductor(0,0), y-c-tipo_conductor)) /* Info al controlador de congestión */ Send (1, y-r-SaleAuto)	Inercial	Sale_ Hacia_ Cruce

Destino = Destino(0,-1) *Camino* = Camino(0,-1) *Velocidad* = VELOCIDAD_INGRESO_CRUCE *Tipo_vehiculo* = tipo_vehiculo (0,-1) *Tipo_conductor* = tipo_conductor (0,-1) *Factor_aceleracion* = 0 Donde Velocidad_Ingreso_Cruce es un parámetro de la simulación	Destino(0,-1) != 0 And Destino(0,0) = 0	/* Info al cruce */ Send(0, y-c-hayauto)	Transport	Llega_ Desde_ Atrás
(0,0)	t /*en otro caso conserva estado */	Send(0, y-c-hayauto)	Transport	Default

El comportamiento de esta celda es exactamente igual al definido en las propuestas anteriores.
En este caso la velocidad del vehículo está dada por un parámetro de la simulación.
El factor de aceleración no se utiliza, por lo tanto, esta celda lo mantiene con estado 0.
El comportamiento al entrar al cruce (o circular por él), es el mismo que el descripto en la primer propuesta (Modelo GAP).

Los modelos correspondientes a los tramos de 2, 3, 4 y más de 4 carriles sufren modificaciones análogas para el caso de un carril.

Definición del modelo que representa a los cruces

Los cruces poseen el mismo comportamiento que el descripto en propuestas anteriores
Se diferencia en los siguientes aspectos:

- El tramo no le informa al cruce la distancia ni la velocidad del auto más cercano hacia adelante, por lo tanto, no posee los ports que comunican esta información.
- Se agrega un port a las celdas de salida del cruce (x-r-peso) que recibe la información de congestión del tramo de salida al cual está conectada. Esto se usa para calcular la velocidad que tendrá el auto cuando ya está listo para salir del cruce.
- La función Velocidad posee otros parámetros, por lo tanto, su definición va a ser distinta.

El port agregado es:

PORTS		
Port	**Nombre**	**Comentario**
Celdas de salida		
$X_{\eta+7}(0,i)$, $i \in$ Out	x-r-peso	Este port se utiliza para conocer la información de congestión correspondiente al tramo de salida.

Acoplamientos

Nota:
- Sólo se especificarán los nuevos acoplamientos, con respecto al modelo presentado en la sección de Ruteo dinámico.

Acoplamiento de los tramos con los cruces

Ports de salida tramo – ingreso cruce

- $Y_{\eta+4}$ (0,k-1) (y-c-tipo_vehiculo) → $X_{\eta+4}$ (Celda de entrada: x-t-tipo_vehiculo) Informando el tipo del vehículo que sale desde el tramo hacia el cruce.
- $Y_{\eta+5}$ (0,k-1) (y-c-tipo_conductor) → $X_{\eta+5}$ (Celda de entrada: x-t-tipo_conductor) Informando el tipo de conductor del auto que sale desde el tramo hacia el cruce.
- $Y_{\eta+6}$ (0,0) (y-c-veloc_maxima) → $X_{\eta+9}$ (Celda de salida: x-t-veloc_maxima) Informando la velocidad máxima permitida en el tramo hacia el cual va a salir el auto desde el cruce.

Ports de salida cruce – ingreso tramo

- $Y_{\eta+6}$ (Celda de salida: y-t-velocidad) → $X_{\eta+3}$ (0,0) (x-c-velocidad) Informando la velocidad del auto que ingresa desde el cruce hacia el tramo.
- $Y_{\eta+4}$ (Celda de salida: y-t-tipo_vehículo) → $X_{\eta+4}$ (0,0) (x-c-tipo_vehiculo) Informando el tipo de vehículo que ingresa desde el cruce hacia el tramo.
- $Y_{\eta+5}$ (Celda de salida: y-t-tipo_conductor) → $X_{\eta+5}$ (0,0) (x-c-tipo_conductor) Informando el tipo de conductor del auto que ingresa desde el cruce hacia el tramo.

Acoplamiento de los tramos con el controlador de congestión

- Y_1 (y-r-peso) → $X_{\eta+5}$ (0,i) (x-r-peso) con $0 \leq i \leq k-2$ (donde k es la cantidad de celdas del tramo): Informando el peso del tramo.

Acoplamiento de los cruces con el controlador de congestión

- Y_1 (y-r-peso) → $X_{\eta+7}$ (Celda de salida: x-r-peso) Informando la velocidad máxima permitida en el tramo hacia el cual va a salir el auto desde el cruce.

Sección V: Conclusiones y trabajos futuros

Conclusiones

En el presente trabajo se ha modelado con éxito un sector real de ciudad utilizando el lenguaje de especificación de alto nivel ATLAS, implementándose el mismo en la herramienta N-CD++. Los resultados obtenidos en las simulaciones de distintos escenarios (hora pico, no pico, etc.) reprodujeron el comportamiento real del sector simulado. Además, con esto se ha demostrado la factibilidad del mapeo del lenguaje de especificación utilizado, con la herramienta de implementación.

Por otra parte, se han agregado a los modelos existentes aspectos no considerados que acercan la simulación al comportamiento real del tráfico, tales como el ruteo de vehículos, el monitoreo de congestión y el comportamiento del conductor.
Mediante los modelos Cell-DEVS se ha logrado la definición de aspectos complejos del comportamiento del tráfico de vehículos, como el ruteo y el monitoreo de congestión, mediante reglas simples, de fácil comprensión, modificación y ejecución eficiente.

Los nuevos modelos de ruteo tanto estático como dinámico permiten, si se tiene información estadística sobre puntos de concentración urbana en una ciudad, modelar el flujo real de vehículos y analizar, por ejemplo, alternativas a implementar en los caminos más transitados. Además, estos modelos mantienen información de congestión permitiendo modelar el comportamiento de los vehículos en diferentes situaciones.

Otro de los aspectos del tráfico analizado fue el comportamiento del conductor. En este caso, los aspectos a modelar, como las características del conductor, las características del vehículo, cuestiones tales como la toma de decisiones ante los cambios de carril, si bien son importantes para caracterizar el movimiento de los vehículos, y es posible definirlos utilizando el formalismo Cell-Devs, habría que analizar cuán efectiva es la simulación considerando el tiempo de cómputo requerido, ya que la complejidad de los modelos definidos hacen suponer que la ejecución de los mismos en sectores de ciudad reales va a requerir un tiempo de procesamiento considerable.

Para probar los modelos definidos se ha utilizado la herramienta N-CD++, la cual ha sido muy útil para definir ejemplos sencillos, pero se hace necesario agregar extensiones que soporten la definición de modelos complejos. Este tema se encuentra detallado en el apéndice C.

De acuerdo con lo expuesto, se concluye que los modelos obtenidos en el presente trabajo para simulación de tráfico urbano, mejora algunos aspectos de los modelos existentes siendo de gran interés y utilidad continuar con este tipo de estudios.

Trabajos Futuros

Se prevee la realización de diversos trabajos en relación con el presente, que serán resumidos a continuación:

- ***Herramienta N-CD++:***
  - ***Posibilidad de definir funciones o macros parametrizables:*** lo que reduciría el tamaño de las reglas, y permitiría una mejor reutilización del código. Por el momento la única forma de incorporar una función es desarrollar un modelo DEVs que implemente la misma. Esto ocurrió con la función que obtiene un nuevo camino, la cual fue implementada mediante el modelo Devs *OD*.

  - ***Nueva extensión del estado de la celda***: En vez de que cada celda pueda almacenar un valor real, sería importante la posibilidad de que también pueda almacenar un arreglo de reales (o una estructura semejante), de tal forma que cada valor represente una variable para la celda, evitando la necesidad de utilizar más dimensiones que las necesarias, como ocurrió en los ejemplos implementados para ruteo estático y dinámico.

  - ***Adecuación del concepto de utilización de los ports que acoplan los modelos***: Este tema se explica en forma detallada en el apéndice C.

  - ***Creación de una interfaz gráfica***: reemplazando la tarea efectuada por *DrawLog* que permite ver los resultados de la simulación de los modelos celulares en modo texto.

- ***Implementación del lenguaje de especificación ATLAS:*** Al implementar los modelos correspondientes al ruteo estático y dinámico se evidenció cuán engorroso y propenso a errores resulta la implementación de modelos complejos, en forma directa en los formalismos Cell-Devs. La implementación del lenguaje de especificación automatizaría el pasaje al formalismo Cell-Devs y facilitaría en gran manera la especificación de sectores de ciudad cuyo estudio resulta interesante.

- ***Comportamiento del conductor:***
  - Si se tuviera implementado el lenguaje de especificación ATLAS, a través del cuál se puede obtener los modelos ejecutables definidos en la sección 4.4 (Comportamiento del conductor), sería interesante realizar un estudio acerca del tiempo de cómputo requerido y la mejora en la reproducción del comportamiento real de los vehículos, de manera de comprobar si el formalismo sigue siendo útil al modelar aspectos más complejos que los implementados en este trabajo.
  - Los modelos presentados en la sección 4.4 pueden ser extendidos para los tramos de varios carriles.
  - Queda pendiente la modelización de los siguientes aspectos del comportamiento del conductor:
    - ***Impaciencia al seguir a un vehículo más lento:*** Influye en la decisión que tomará el conductor cuando se le presente la posibilidad de cambiar de carril.
    - ***Distancia crítica para cambiar de carril:*** Influye en la decisión que tomará el conductor cuando se le presente la posibilidad de cambiar de carril.

-

Sección VI: Apéndices

Apéndice A: Modelos DEVs implementados

El objetivo de esta sección es describir los modelos DEVs Congestion y OD codificados en C++ y utilizados para implementar el ruteo estático, dinámico y el uso de matrices origen destino.

Modelo Congestión

El modelo Congestión se utiliza para medir cuan congestionado está un tramo, analizando la cantidad de autos que tiene el mismo en un determinado momento. Para poder realizar esto el modelo Congestión cuenta con dos ports de entrada: uno para informar la entrada de un auto al tramo y el otro para informar la salida de un auto del tramo. De este modo el Modelo sabe en todo momento cuantos autos tiene un tramo, pudiendo consultarsele esta información a travez de un port (peso).

```
congestion::congestion( const string &name)
            :Atomic(name)
            , IngresaAuto(this->addInputPort("IngresaAuto"))
            , SaleAuto(this->addInputPort("SaleAuto"))
            , PreguntaPeso(this->addInputPort("PreguntaPeso"))
            , stop(this->addInputPort("stop"))
            , peso(this->addOutputPort("peso"))
{
  if( MainSimulator::Instance().existsParameter( description(), "autos" ) )
            cant_autos = str2Int(MainSimulator::Instance().getParameter( description(), "autos" ) );

}
//*********************************************************************
Model &congestion::initFunction()
{
   return *this;
}
//*********************************************************************
Model &congestion::externalFunction(const ExternalMessage &msg)
{
 if (msg.port() == IngresaAuto )
  {
    cant_autos++;
  }
  if (msg.port() == SaleAuto )
  {
    cant_autos--;
  }
  if (msg.port() == PreguntaPeso )
  {
    holdIn( active, Time::Zero );
  }
  if (msg.port() == stop )
  {
```

```
            this->passivate();
  }
  return *this;
}
//*********************************************************************
Model &congestion::outputFunction( const InternalMessage &msg)
{
      sendOutput(msg.time(), peso, cant_autos );
      return *this ;
}
//*********************************************************************
Model &congestion::internalFunction (const InternalMessage &msg)
{
            passivate();
            return *this;
}
```

Modelo OD

El modelo OD básicamente administra una matriz origen – destino. Esta información puede ser utilizada por un auto para obtener un camino alternativo entre un par origen – destino determinado. El modelo OD requiere una inicialización previa, que se realiza a través de un archivo (od.in) con el siguiente formato:

$Origen_1$ # $destino_1$ # $camino_1$
$Origen_2$ # $destino_2$ # $camino_2$
.....
$origen_n$ # $destino_n$ # $camino_n$

Este modelo se desarrolló para implementar la función Nuevo_Camino (Ver apéndice E).
El mismo permite que a través de un port, se le solicite un camino alternativo, informando el origen, el destino y el camino actual. Así, busca en la matriz OD el primer camino cuyo origen y destino sean los informados a través del port y que difiera del camino actual.

```
Od::od( const string &name)
            :Atomic(name)
            , SolicitaCamino(this->addInputPort("SolicitaCamino"))
            , stop(this->addInputPort("stop"))
            , InformaCamino(this->addOutputPort("InformaCamino"))
{
}
//*********************************************************************
od::~od()
{
   matriz.erase( matriz.begin(), matriz.end() ) ;
}
//*********************************************************************
Model &od::initFunction()
{
   IniciarMatriz();
```

```
    camino = 0;
    return *this;
}
//*********************************************************************
Model &od::externalFunction(const ExternalMessage &msg)
{
  /*******************************************************************
   *  En el mensaje recibo el origen, el destino y el proximo tramo                *
   *  de la siguiente forma : T.OOODDD, donde                                      *
   *  T: proximo tramo                                                             *
   *  OOO: origen, con ceros a la izquierda de relleno                             *
   *  DDD: destino, con ceros a la izquierda de relleno                             *
   *******************************************************************/

  list<int> l_camino;
  double l_camino_format = 0;
  int l_origen =0;
  int l_destino =0;
  int l_proximoTramo =0;
  double l_aux = 0;
  double l_value = 0;

  if (msg.port() == SolicitaCamino )
  {
    l_value = msg.value();
    l_proximoTramo = floor(l_value);
    l_aux = ( l_value - floor(l_value) ) * pow(10,2); // OOO,DDD
    l_origen = floor(l_aux);  // OOO
    l_aux = (l_aux - floor(l_aux)) * pow(10,2);

    // Redondeo
    if ((l_aux - floor(l_aux) ) > 0.5)
            l_destino = ceil(l_aux);
    else
            l_destino = floor(l_aux);

    l_camino = ObtenerCamino( l_proximoTramo, l_origen, l_destino);
    camino =ConvertirCamino(l_camino);
    holdIn( active, Time::Zero );
  }

  if (msg.port() == stop )
  {
          this->passivate();
  }
  return *this;
}
//*********************************************************************
Model &od::outputFunction( const InternalMessage &msg)
{
      sendOutput(msg.time(), InformaCamino, camino);
      return *this ;
}
//*********************************************************************
Model &od::internalFunction (const InternalMessage &msg)
{
```

```
        passivate();
        return *this;
}

//*********************************************************************
od::IniciarMatriz ()
{
    matrizOD elem;
    string inputFile("od.in");
    FILE *fileIn;
     char line[256];
     char l_origen[10];
     char l_destino[10] ;
     char l_tramo[10];
     char *endptr;
     int i = 0;
     int j = 0;
     int t = 0;
     long long_tramo = 0;

    MASSERTMSG(fileIn = fopen (inputFile.c_str(), "r"), "Can't open the input file \n");
    while (fgets (line, 255, fileIn)!= NULL)
        {
                        i=0;
                        printf("linea= %s\n", line);
                        while(! isdigit(line[i]))
                        i++;

                        // origen
                        j=0;
                        while(isdigit(line[i]))
                        {
                                l_origen[j]= line[i];
                                j++;
                                i++;
                        }

                        while(! isdigit(line[i]))
                                i++;
                        // destino
                        j=0;
                        while(isdigit(line[i]))
                        {
                                l_destino[j]= line[i];
                                j++;
                                i++;
                        }

                        // camino
                        elem.camino.erase(elem.camino.begin(), elem.camino.end());
                        while((line[i])!= '\n')
                        {
                                while(! isdigit(line[i]))
                                        i++;

                                //Encuentro un tramo del camino
```

```
                                        t=0;
                                        while (isdigit(line[i]))
                                        {
                                                l_tramo[t] = line[i];
                                                t++;
                                                i++;
                                        }
                                        l_tramo[t]= "\n";
                                        long_tramo = strtol(l_tramo, &endptr, 0);
                                        elem.camino.push_back(long_tramo);
                                                        }

                                elem.origen= strtol(l_origen, &endptr,0);
                                elem.destino= strtol(l_destino, &endptr,0);
                                matriz.push_back(elem);
        }
        fclose(fileIn);

}
//**********************************************************************
list<int> od::ObtenerCamino(int pproximo_tramo, int porigen,int pdestino)
{
        Camino l_camino;

        list<matrizOD>::const_iterator cursor( matriz.begin() ) ;

        for( ; cursor != matriz.end() ; cursor ++ )
        {
                list<int>::const_iterator cam_cur ( ((*cursor).camino).begin() );

                if ( ( (*cursor).origen == porigen )  &&
                    ( (*cursor).destino == pdestino ) &&
                    ( *cam_cur  != pproximo_tramo) )
                {

                l_camino =   (*cursor).camino;
                }

        }
        return (l_camino);
}

//**********************************************************************
double od::ConvertirCamino(list<int> p_camino)
{
        double l_camino = 0;
        int i=0;
        list<int>::const_iterator cursor(p_camino.begin() ) ;

        for( ; cursor != p_camino.end() ; ++cursor)
        {
                i++;
                l_camino = ((*cursor) * pow(10, -(i-1)) ) + l_camino;

        }
        return (l_camino);
```

```
}
```

Apéndice B: Especificaciones en N-CD++

1.1. Ejemplo de ruteo estático

1.1.1. Archivo : re.ma

```
#include(re.inc)

[top]
components : tramo1 tramo2 tramo3 tramo4 tramo5 cruce1 cruce2 cruce3 cruce4

in : inNuevoAuto inNuevoCamino
%* Puertos de entrada al modelo acoplado
link : inNuevoAuto inAuto@tramo1
link : inNuevoCamino inCamino@tramo1
% Acoplamiento Tramo1-Cruce1
link : outAuto@Tramo1 inAuto@Cruce1
link : outCamino@Tramo1 inCamino@Cruce1
link : outHayLugarParaEntrar@Cruce1 inHayLugar@Tramo1
link : outQuiereCruzar@Tramo1 inHayAuto@Cruce1
% Acoplamiento Cruce1-Tramo2
link : outAuto2@Cruce1 inAuto@Tramo2
link : outCamino2@Cruce1 inCamino@Tramo2
link : outPuedeSalir2@Cruce1 inPuedeEntrar@Tramo2
link : outPuedeEntrar@Tramo2 inHayLugar2@Cruce1
% Acoplamiento Cruce1-Tramo3
link : outAuto3@Cruce1 inAuto@Tramo3
link : outCamino3@Cruce1 inCamino@Tramo3
link : outPuedeSalir3@Cruce1 inPuedeEntrar@Tramo3
link : outPuedeEntrar@Tramo3 inHayLugar3@Cruce1
% Acoplamiento Tramo2-Cruce2
link : outAuto@Tramo2 inAuto@Cruce2
link : outCamino@Tramo2 inCamino@Cruce2
link : outHayLugarParaEntrar@Cruce2 inHayLugar@Tramo2
link : outQuiereCruzar@Tramo2 inHayAuto@Cruce2
% Acoplamiento Tramo3-Cruce3
link : outAuto@Tramo3 inAuto@Cruce3
link : outCamino@Tramo3 inCamino@Cruce3
link : outHayLugarParaEntrar@Cruce3 inHayLugar@Tramo3
link : outQuiereCruzar@Tramo3 inHayAuto@Cruce3
% Acoplamiento Cruce2-Tramo5
link : outAuto5@Cruce2 inAuto@Tramo5
link : outCamino5@Cruce2 inCamino@Tramo5
link : outPuedeSalir5@Cruce2 inPuedeEntrar@Tramo5
link : outPuedeEntrar@Tramo5 inHayLugar5@Cruce2
% Acoplamiento Cruce3-Tramo4
link : outAuto4@Cruce3 inAuto@Tramo4
```

```
link : outCamino4@Cruce3 inCamino@Tramo4
link : outPuedeSalir4@Cruce3 inPuedeEntrar@Tramo4
link : outPuedeEntrar@Tramo4 inHayLugar4@Cruce3
% Acoplamiento Tramo4-Cruce4
link : outAuto@Tramo4 inAuto@Cruce4
link : outCamino@Tramo4 inCamino@Cruce4
link : outHayLugarParaEntrar@Cruce4 inHayLugar@Tramo4
link : outQuiereCruzar@Tramo4 inHayAuto@Cruce4
% Acoplamiento Tramo5-Cruce4
link : outAuto@Tramo5 inAuto@Cruce4
link : outCamino@Tramo5 inCamino@Cruce4
link : outHayLugarParaEntrar@Cruce4 inHayLugar@Tramo5
link : outQuiereCruzar@Tramo5 inHayAuto@Cruce4

[tramo1]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo1(0,-1)  tramo1(0,0)  tramo1(0,1)
neighbors : tramo1(1,-1)  tramo1(1,0)  tramo1(1,1)
initialvalue : 0
initialrow : 0     0 0 0 1.0 0
initialrow : 1     0 0 0 2.5 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar
link : inAuto inAuto@Tramo1(0,0)
link : inCamino inCamino@Tramo1(1,0)
link : inHayLugar inHayLugar@Tramo1(0,4)
link : inHayLugar inHayLugar@Tramo1(1,4)
out : outQuiereCruzar outAuto outCamino
link : outQuiereCruzar@tramo1(0,4) outQuiereCruzar
link : outAuto@tramo1(0,4) outAuto
link : outCamino@tramo1(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inAuto@tramo1(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo1(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo1(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo1(1,4) SaleACruceCamino

[tramo2]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo2(0,-1)  tramo2(0,0)  tramo2(0,1)
neighbors : tramo2(1,-1)  tramo2(1,0)  tramo2(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
link : inPuedeEntrar inPuedeEntrar@Tramo2(0,0)
link : inAuto inAuto@Tramo2(0,0)
```

```
link : inCamino inCamino@Tramo2(1,0)
link : inHayLugar inHayLugar@Tramo2(0,4)
link : inHayLugar inHayLugar@Tramo2(1,4)
out : outQuiereCruzar outAuto outCamino outPuedeEntrar
link : outPuedeEntrar@tramo2(0,0) outPuedeEntrar
link : outQuiereCruzar@tramo2(0,4) outQuiereCruzar
link : outAuto@tramo2(0,4) outAuto
link : outCamino@tramo2(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inPuedeEntrar@tramo2(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo2(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo2(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo2(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo2(1,4) SaleACruceCamino

[tramo3]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo3(0,-1)  tramo3(0,0)  tramo3(0,1)
neighbors : tramo3(1,-1)  tramo3(1,0)  tramo3(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
link : inPuedeEntrar inPuedeEntrar@Tramo3(0,0)
link : inAuto inAuto@Tramo3(0,0)
link : inCamino inCamino@Tramo3(1,0)
link : inHayLugar inHayLugar@Tramo3(0,4)
link : inHayLugar inHayLugar@Tramo3(1,4)
out  : outQuiereCruzar outAuto outCamino outPuedeEntrar
link : outPuedeEntrar@tramo3(0,0) outPuedeEntrar
link : outQuiereCruzar@tramo3(0,4) outQuiereCruzar
link : outAuto@tramo3(0,4) outAuto
link : outCamino@tramo3(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inPuedeEntrar@tramo3(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo3(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo3(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo3(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo3(1,4) SaleACruceCamino

[tramo4]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo4(0,-1)  tramo4(0,0)  tramo4(0,1)
neighbors : tramo4(1,-1)  tramo4(1,0)  tramo4(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
```

```
link : inPuedeEntrar inPuedeEntrar@Tramo4(0,0)
link : inAuto inAuto@Tramo4(0,0)
link : inCamino inCamino@Tramo4(1,0)
link : inHayLugar inHayLugar@Tramo4(0,4)
link : inHayLugar inHayLugar@Tramo4(1,4)
out  : outQuiereCruzar outAuto outCamino outPuedeEntrar
link : outPuedeEntrar@tramo4(0,0) outPuedeEntrar
link : outQuiereCruzar@tramo4(0,4) outQuiereCruzar
link : outAuto@tramo4(0,4) outAuto
link : outCamino@tramo4(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inPuedeEntrar@tramo4(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo4(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo4(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo4(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo4(1,4) SaleACruceCamino

[tramo5]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo5(0,-1)  tramo5(0,0)  tramo5(0,1)
neighbors : tramo5(1,-1)  tramo5(1,0)  tramo5(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
link : inPuedeEntrar inPuedeEntrar@Tramo5(0,0)
link : inAuto inAuto@Tramo5(0,0)
link : inCamino inCamino@Tramo5(1,0)
link : inHayLugar inHayLugar@Tramo5(0,4)
link : inHayLugar inHayLugar@Tramo5(1,4)
out  : outQuiereCruzar outAuto outCamino outPuedeEntrar
link : outPuedeEntrar@tramo5(0,0) outPuedeEntrar
link : outQuiereCruzar@tramo5(0,4) outQuiereCruzar
link : outAuto@tramo5(0,4) outAuto
link : outCamino@tramo5(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inPuedeEntrar@tramo5(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo5(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo5(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo5(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo5(1,4) SaleACruceCamino

[PuedeEntrarTramo]
% Le indica al cruce si hay lugar para que ingrese el auto
rule : { (0,0) + send(outPuedeEntrar,(0,0)) } 10 { t }

[IngresaAutoTramo]
% Ingresa un auto al tramo (posiblemente desde un cruce)
rule : 1 10 { portvalue(ThisPort) = 1 and (0,0) = 0 }
rule : 0 10  { t }

[IngresaCaminoTramo]
```

```
rule : {portvalue(ThisPort)} 10 {portvalue(ThisPort) != 0 and (0,0) = 0}
rule : 0 10  { t }

[tramo-rule]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) = 1 and (0,1) = 0 }
% Viene de atras
rule : 1 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) = 1 }
% Sale hacia cruce
rule : {1 + send(outQuiereCruzar, 1)} 10 { not isundefined((0,-1)) and isundefined((0,1)) and  (0,0) = 1 }
rule : {(0,0)} 10  { t }

[Caminotramo-rule]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) != 0 and (0,1) = 0 }
% Viene de atras
rule : {(0,-1)} 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) != 0 }
rule : {(0,0)} 10  { t }

[SaleACruceAuto]
% Sale hacia adelante (para la celda de salida)
rule : { 0 + send(outAuto,1) } 10 { (0,0) = 1 and portvalue(ThisPort) = 0 }
rule : {(0,0)} 10  { t }

[SaleACruceCamino]
% Sale hacia adelante (para la celda de salida)
rule : { 0 + send(outCamino,(0,0)) } 10 { (0,0) != 0 and portvalue(ThisPort) = 0 }
rule : {(0,0)} 10  { t }

[cruce1]
type : cell
Width : 3
Height : 4
delay : transport
defaultDelayTime  : 50
border : wrapped
neighbors : cruce1(0,-1) cruce1(0,0) cruce1(0,1)
neighbors : cruce1(1,-1) cruce1(1,0) cruce1(1,1)
neighbors : cruce1(2,-1) cruce1(2,0) cruce1(2,1)
neighbors : cruce1(3,-1) cruce1(3,0) cruce1(3,1)
initialvalue : 0
localtransition : celdas-entrada
in : inAuto inCamino inHayAuto inHayLugar2 inHayLugar3
link : inAuto inAuto@Cruce1(0,0)
link : inCamino inCamino@Cruce1(1,0)
link : inHayAuto inPregunta@Cruce1(0,0)
link : inHayLugar2 inHayLugar@Cruce1(0,1)
link : inHayLugar2 inHayLugar@Cruce1(1,1)
link : inHayLugar3 inHayLugar@Cruce1(0,2)
link : inHayLugar3 inHayLugar@Cruce1(1,2)
out : outHayLugarParaEntrar outPuedeSalir2 outAuto2 outCamino2 outPuedeSalir3 outAuto3 outCamino3
link : haylugar@cruce1(0,0) outHayLugarParaEntrar
link : PuedeSalir@cruce1(0,1) outPuedeSalir2
link : auto@cruce1(0,1) outAuto2
link : camino@cruce1(1,1) outCamino2
link : PuedeSalir@cruce1(0,2) outPuedeSalir3
```

```
link : auto@cruce1(0,2) outAuto3
link : camino@cruce1(1,2) outCamino3
initialrowvalue : 2     111
initialrowvalue : 3     023
zone : celdas-salida { (0,1)..(0,2) }
zone : CaminoCruce-entrada { (1,0) }
zone : CaminoCruce-salida { (1,1)..(1,2) }
zone : idcruce-rule { (2,0)..(2,2) }
zone : idtramo-rule { (3,0)..(3,2) }
portInTransition : inAuto@cruce1(0,0) IngresaAutoCruce
portInTransition : inCamino@cruce1(1,0) IngresaCaminoCruce
portInTransition : inPregunta@cruce1(0,0) QuiereCruzar
portInTransition : inHayLugar@cruce1(0,1) PuedeSalirAuto
portInTransition : inHayLugar@cruce1(0,2) PuedeSalirAuto
portInTransition : inHayLugar@cruce1(1,1) PuedeSalirCamino
portInTransition : inHayLugar@cruce1(1,2) PuedeSalirCamino

[cruce2]
type : cell
Width : 2
Height : 4
delay : transport
defaultDelayTime  : 50
border : wrapped
neighbors : cruce2(0,-1) cruce2(0,0) cruce2(0,1)
neighbors : cruce2(1,-1) cruce2(1,0) cruce2(1,1)
neighbors : cruce2(2,-1) cruce2(2,0) cruce2(2,1)
neighbors : cruce2(3,-1) cruce2(3,0) cruce2(3,1)
initialvalue : 0
localtransition : celdas-entrada
in : inAuto inCamino inHayAuto inHayLugar5
link : inAuto inAuto@Cruce2(0,0)
link : inCamino inCamino@Cruce2(1,0)
link : inHayAuto inPregunta@Cruce2(0,0)
link : inHayLugar5 inHayLugar@Cruce2(0,1)
link : inHayLugar5 inHayLugar@Cruce2(1,1)
out : outHayLugarParaEntrar outPuedeSalir5 outAuto5 outCamino5
link : haylugar@cruce2(0,0) outHayLugarParaEntrar
link : PuedeSalir@cruce2(0,1) outPuedeSalir5
link : auto@cruce2(0,1) outAuto5
link : camino@cruce2(1,1) outCamino5
initialrowvalue : 2     22
initialrowvalue : 3     05
zone : celdas-salida { (0,1) }
zone : CaminoCruce-entrada { (1,0) }
zone : CaminoCruce-salida { (1,1) }
zone : idcruce-rule { (2,0)..(2,1) }
zone : idtramo-rule { (3,0)..(3,1) }
portInTransition : inAuto@cruce2(0,0) IngresaAutoCruce
portInTransition : inCamino@cruce2(1,0) IngresaCaminoCruce
portInTransition : inPregunta@cruce2(0,0) QuiereCruzar
portInTransition : inHayLugar@cruce2(0,1) PuedeSalirAuto
portInTransition : inHayLugar@cruce2(1,1) PuedeSalirCamino

[cruce3]
type : cell
```

```
Width : 2
Height : 4
delay : transport
defaultDelayTime  : 50
border : wrapped
neighbors : cruce3(0,-1) cruce3(0,0) cruce3(0,1)
neighbors : cruce3(1,-1) cruce3(1,0) cruce3(1,1)
neighbors : cruce3(2,-1) cruce3(2,0) cruce3(2,1)
neighbors : cruce3(3,-1) cruce3(3,0) cruce3(3,1)
initialvalue : 0
localtransition : celdas-entrada
in : inAuto inCamino inHayAuto inHayLugar4
link : inAuto inAuto@Cruce3(0,0)
link : inCamino inCamino@Cruce3(1,0)
link : inHayAuto inPregunta@Cruce3(0,0)
link : inHayLugar4 inHayLugar@Cruce3(0,1)
link : inHayLugar4 inHayLugar@Cruce3(1,1)
out : outHayLugarParaEntrar outPuedeSalir4 outAuto4 outCamino4
link : haylugar@cruce3(0,0) outHayLugarParaEntrar
link : PuedeSalir@cruce3(0,1) outPuedeSalir4
link : auto@cruce3(0,1) outAuto4
link : camino@cruce3(1,1) outCamino4
initialrowvalue : 2     33
initialrowvalue : 3     04
zone : celdas-salida { (0,1) }
zone : CaminoCruce-entrada { (1,0) }
zone : CaminoCruce-salida { (1,1) }
zone : idcruce-rule { (2,0)..(2,1) }
zone : idtramo-rule { (3,0)..(3,1) }
portInTransition : inAuto@cruce3(0,0) IngresaAutoCruce
portInTransition : inCamino@cruce3(1,0) IngresaCaminoCruce
portInTransition : inPregunta@cruce3(0,0) QuiereCruzar
portInTransition : inHayLugar@cruce3(0,1) PuedeSalirAuto
portInTransition : inHayLugar@cruce3(1,1) PuedeSalirCamino

[cruce4]
type : cell
Width : 2
Height : 4
delay : transport
defaultDelayTime  : 50
border : wrapped
neighbors : cruce4(0,-1) cruce4(0,0)  cruce4(0,1)
neighbors : cruce4(1,-1) cruce4(1,0)  cruce4(1,1)
neighbors : cruce4(2,-1) cruce4(2,0)  cruce4(2,1)
neighbors : cruce4(3,-1) cruce4(3,0)  cruce4(3,1)
initialvalue : 0
localtransition : celdas-entrada
in : inAuto inCamino inHayAuto inHayLugar
link : inAuto inAuto@Cruce4(0,0)
link : inCamino inCamino@Cruce4(1,0)
link : inHayAuto inPregunta@Cruce4(0,0)
link : inHayLugar inHayLugar@Cruce4(0,1)
link : inHayLugar inHayLugar@Cruce4(1,1)
out : outHayLugarParaEntrar outPuedeSalir outAuto outCamino
link : haylugar@cruce4(0,0) outHayLugarParaEntrar
```

```
link : PuedeSalir@cruce4(0,1) outPuedeSalir
link : auto@cruce4(0,1) outAuto
link : camino@cruce4(1,1) outCamino
initialrowvalue : 2     44
initialrowvalue : 3     00
zone : celdas-salida { (0,1) }
zone : CaminoCruce-entrada { (1,0) }
zone : CaminoCruce-salida { (1,1) }
zone : idcruce-rule { (2,0)..(2,1) }
zone : idtramo-rule { (3,0)..(3,1) }
portInTransition : inAuto@cruce4(0,0) IngresaAutoCruce
portInTransition : inCamino@cruce4(1,0) IngresaCaminoCruce
portInTransition : inPregunta@cruce4(0,0) QuiereCruzar
portInTransition : inHayLugar@cruce4(0,1) PuedeSalirAuto
portInTransition : inHayLugar@cruce4(1,1) PuedeSalirCamino

[IngresaAutoCruce]
rule : 1 10 { (0,0) = 0 and portvalue(ThisPort) = 1 }
rule : {(0,0)} 10  { t }

[IngresaCaminoCruce]
rule : {portvalue(ThisPort)} 10 { (0,0) = 0 and portvalue(ThisPort) != 0 }
rule : {(0,0)} 10  { t }

[QuiereCruzar]
rule : { (0,0) + send(haylugar,0) } 10 { (0,0) = 0 and (0,-1) = 0 and portvalue(ThisPort) = 1 }
rule : { (0,0) + send(HayLugar,1) } 10  { ((0,0) = 1 or ((0,0) = 0 and (0,-1) = 1)) and portvalue(ThisPort) = 1 }
rule : {(0,0)} 10  { t }

[celdas-entrada]
% Elimina el auto de la simulación si ha llegado a destino
rule : 0 10 { (0,0) = 1 and #Macro(Es_Destino) }
% Ingresa a la celda un auto que ya estaba dentro del cruce
rule : 1 10 { (0,0) = 0 and (0,-1) = 1 and (1,-1) != -1 }
% Avanza a la prox celda
rule : 0 10 { (0,0) = 1 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[celdas-salida]
% Avanza el auto si no debe salir por el tramo (de acuerdo al camino)
rule : 1 10 { (0,0) = 0 and (0,-1) = 1 and not #Macro(SalirAuto) }
% Le indica al tramo que hay un auto que puede entrar al mismo (si corresponde al camino)
rule : { 0 + send(PuedeSalir,1)} 10 { (0,0) = 0 and (0,-1) = 1 and #Macro(SalirAuto) }
% Avance de un auto dentro del cruce
rule : 0 10 { (0,0) = 1 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[CaminoCruce-entrada]
% Ingresa a la celda un camino que ya estaba dentro del cruce
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) > 0 }
% Elimina el camino de la simulación si ha llegado a destino
rule : 0 10 { (0,0) != 0 and #Macro(Es_Destino) }
% Avanza a la prox celda
%rule : 0 10 { (0,0) != 0 and (0,1) = 0 }
rule : 0 10 { (0,0) != 0 }
rule : {(0,0)} 10  { t }
```

```
[CaminoCruce-salida]
% Avanza el camino si no debe salir por el tramo (de acuerdo al camino)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) != 0 and not #Macro(SalirCamino) }
% Avance de un camino dentro del cruce
rule : 0 10 { (0,0) != 0 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[idcruce-rule]
rule : {(0,0)} 10  { t }

[idtramo-rule]
rule : {(0,0)} 10  { t }

[PuedeSalirAuto]
% Llega auto que abandonara el cruce
rule : {0 + send(auto, 1)} 10 { (0,0) = 0 and (0,-1) = 1 and portvalue(ThisPort) = 0 and #Macro(SalirAuto) }
% Llega auto que permanecera dentro del cruce
rule : {1 + send(auto, 0)} 10 { (0,0) = 0 and (0,-1) = 1 and ( portvalue(ThisPort) = 1 or ( portvalue(ThisPort) =
0 and not #Macro(SalirAuto) )) }
rule : {(0,0)} 10  { t }

[PuedeSalirCamino]
% Llega auto que abandonara el cruce (no es el último tramo)
% Nota: se reemplazó la condición #Macro(NuevoCamino) != 0 por #Macro(NuevoCamino) >= 0.0001 por
problemas de redondeo.
% Esta condición funciona siempre y cuando la identificación de los tramos tenga hasta 4 dígitos.
rule : {0 + send(camino,#Macro(NuevoCamino))} 10 { (0,0) = 0 and (0,-1) != 0 and portvalue(ThisPort) = 0
and #Macro(SalirCamino) and #Macro(NuevoCamino) >= 0.0001}
% Llega auto que abandonara el cruce (es el último tramo)
rule : {0 + send(camino,-1)} 10 { (0,0) = 0 and (0,-1) != 0 and portvalue(ThisPort) = 0 and
#Macro(SalirCamino) and #Macro(NuevoCamino) < 0.0001}
% Llega camino que permanecera dentro del cruce
rule : {(0,-1) + send(camino, 0)} 10 { (0,0) = 0 and (0,-1) != 0 and ( portvalue(ThisPort) = 1 or (
portvalue(ThisPort) = 0 and not #Macro(SalirCamino) )) }
rule : {(0,0)} 10  { t }
```

1.1.2. **Archivo : re.inc**

```
#BeginMacro(SalirAuto)
 (Trunc((1,-1)) = (3,0) )
#EndMacro

#BeginMacro(SalirCamino)
 (Trunc((0,-1)) = (2,0) )
#EndMacro

#BeginMacro(Es_destino)
 ( (1,0) = -1 )
#EndMacro

#BeginMacro(NuevoCamino)
 ( Fractional((0,-1)) * 10 )
```

```
#EndMacro
```

1.2. Ejemplo de sector de ciudad

1.2.1. Archivo : ciudad.ma

```
#include(ciudad.inc)

[top]
components : rA rB rC rE rF rD1 rD2 rG1 rG2 rH1 rH2 rI1 rI2 via c1 c2 c3 c4  estacion@Generator
components : genAuto1@generator genAuto2@generator genAuto3@generator genAuto4@generator
components : cont1@contador
Out : outTrans1 outTrans2

link :  throughputS@cont1 outTrans2
link : throughputA@cont1 outTrans1

link : out@genAuto3 inAuto@rA
link : out@genAuto3 arrived@cont1
link : out@genAuto1 inAuto0@rG1
link : out@genAuto1 arrived@cont1
link : out@genAuto1 inAuto1@rG1
link : out@genAuto1 arrived@cont1
link : out@genAuto1 inAuto2@rG1
link : out@genAuto1 arrived@cont1
link : out@genAuto1 inAuto3@rG1
link : out@genAuto2 arrived@cont1
link : out@genAuto2 inAuto0@rH2
link : out@genAuto2 arrived@cont1
link : out@genAuto2 inAuto1@rH2
link : out@genAuto2 arrived@cont1
link : out@genAuto4 inAuto0@rI2
link : out@genAuto4 arrived@cont1
link : out@genAuto4 inAuto1@rI2
link : out@genAuto4 arrived@cont1

link : outAuto0@rG2 solved@cont1
link : outAuto1@rG2 solved@cont1
link : outAuto2@rG2 solved@cont1
link : outAuto3@rG2 solved@cont1
link : outAuto0@rH1 solved@cont1
link : outAuto1@rH1 solved@cont1
link : outAuto0@rI1 solved@cont1
link : outAuto1@rI1 solved@cont1
link : outAuto@rF  solved@cont1

% Acoplamiento via - tramo rI1
link : outPasaTren2@via inPasaTrenAntes0@rI1
link : outPasaTren2@via inPasaTrenDespues0@rI1
link : outPasaTren3@via inPasaTrenAntes1@rI1
link : outPasaTren3@via inPasaTrenDespues1@rI1
link : outPasaTren0@rI1 inPasaTren2@via
link : outPasaTren1@rI1 inPasaTren3@via

% Acoplamiento via - tramo rI2
link : outPasaTren1@via inPasaTrenAntes0@rI2
link : outPasaTren1@via inPasaTrenDespues0@rI2
```

```
link : outPasaTren0@via inPasaTrenAntes1@rI2
link : outPasaTren0@via inPasaTrenDespues1@rI2
link : outPasaTren0@rI2 inPasaTren1@via
link : outPasaTren1@rI2 inPasaTren0@via

% Acoplamiento via - estacion
link : out@Estacion inTren@via


%******** ACOPLAMIENTOS CRUCE c1 ***************************
% Acoplamiento entrada al cruce rG1-c1
link : outAuto0@rG1 inAuto4@c1
link : outAuto1@rG1 inAuto5@c1
link : outAuto2@rG1 inAuto6@c1
link : outAuto3@rG1 inAuto7@c1
link : outHayLugar4@c1 inHayLugar0@rG1
link : outHayLugar5@c1 inHayLugar1@rG1
link : outHayLugar6@c1 inHayLugar2@rG1
link : outHayLugar7@c1 inHayLugar3@rG1
link : outQuiereCruzar0@rG1 inHayAuto4@c1
link : outQuiereCruzar1@rG1 inHayAuto5@c1
link : outQuiereCruzar2@rG1 inHayAuto6@c1
link : outQuiereCruzar3@rG1 inHayAuto7@c1

% Acoplamiento entrada al cruce rD1-c1
link : outAuto0@rD1 inAuto10@c1
link : outAuto1@rD1 inAuto11@c1
link : outHayLugar10@c1 inHayLugar0@rD1
link : outHayLugar11@c1 inHayLugar1@rD1
link : outQuiereCruzar0@rD1 inHayAuto10@c1
link : outQuiereCruzar1@rD1 inHayAuto11@c1

% Acoplamiento entrada al cruce rH2-c1
link : outAuto0@rH2 inAuto15@c1
link : outAuto1@rH2 inAuto16@c1
link : outHayLugar15@c1 inHayLugar0@rH2
link : outHayLugar16@c1 inHayLugar1@rH2
link : outQuiereCruzar0@rH2 inHayAuto15@c1
link : outQuiereCruzar1@rH2 inHayAuto16@c1

% Acoplamiento entrada al cruce rI2-c1
link : outAuto0@rI2 inAuto19@c1
link : outAuto1@rI2 inAuto20@c1
link : outHayLugar19@c1 inHayLugar0@rI2
link : outHayLugar20@c1 inHayLugar1@rI2
link : outQuiereCruzar0@rI2 inHayAuto19@c1
link : outQuiereCruzar1@rI2 inHayAuto20@c1

% Acoplamiento salida del cruce c1-rD2
link : outAuto8@c1 inAuto0@rD2
link : outAuto9@c1 inAuto1@rD2
link : outPuedeSalir8@c1 inPuedeEntrar0@rD2
link : outPuedeSalir9@c1 inPuedeEntrar1@rD2
link : outPuedeEntrar0@rD2 inHayLugar8@c1
link : outPuedeEntrar1@rD2 inHayLugar9@c1
```

```
% Acoplamiento salida del cruce c1-rE
link : outAuto12@c1 inAuto@rE
link : outPuedeSalir12@c1 inPuedeEntrar@rE
link : outPuedeEntrar@rE inHayLugar12@c1

% Acoplamiento salida del cruce c1-rG2
link : outAuto0@c1 inAuto3@rG2
link : outAuto1@c1 inAuto2@rG2
link : outAuto2@c1 inAuto1@rG2
link : outAuto3@c1 inAuto0@rG2
link : outPuedeSalir0@c1 inPuedeEntrar3@rG2
link : outPuedeSalir1@c1 inPuedeEntrar2@rG2
link : outPuedeSalir2@c1 inPuedeEntrar1@rG2
link : outPuedeSalir3@c1 inPuedeEntrar0@rG2
link : outPuedeEntrar3@rG2 inHayLugar0@c1
link : outPuedeEntrar2@rG2 inHayLugar1@c1
link : outPuedeEntrar1@rG2 inHayLugar2@c1
link : outPuedeEntrar0@rG2 inHayLugar3@c1

% Acoplamiento salida del cruce c1-rH1
link : outAuto14@c1 inAuto0@rH1
link : outAuto13@c1 inAuto1@rH1
link : outPuedeSalir14@c1 inPuedeEntrar0@rH1
link : outPuedeSalir13@c1 inPuedeEntrar1@rH1
link : outPuedeEntrar0@rH1 inHayLugar14@c1
link : outPuedeEntrar1@rH1 inHayLugar13@c1

% Acoplamiento salida del cruce c1-rI1
link : outAuto18@c1 inAuto0@rI1
link : outAuto17@c1 inAuto1@rI1
link : outPuedeSalir18@c1 inPuedeEntrar0@rI1
link : outPuedeSalir17@c1 inPuedeEntrar1@rI1
link : outPuedeEntrar0@rI1 inHayLugar18@c1
link : outPuedeEntrar1@rI1 inHayLugar17@c1

%******** ACOPLAMIENTOS CRUCE c2 ***************************
% Acoplamiento entrada al cruce rA-c2
link : outAuto@rA inAuto0@c2
link : outHayLugar0@c2 inHayLugar@rA
link : outQuiereCruzar@rA inHayAuto0@c2

% Acoplamiento entrada al cruce rD2-c2
link : outAuto0@rD2 inAuto4@c2
link : outAuto1@rD2 inAuto5@c2
link : outHayLugar4@c2 inHayLugar0@rD2
link : outHayLugar5@c2 inHayLugar1@rD2
link : outQuiereCruzar0@rD2 inHayAuto4@c2
link : outQuiereCruzar1@rD2 inHayAuto5@c2

% Acoplamiento salida del cruce c2-rB
link : outAuto1@c2 inAuto@rB
link : outPuedeSalir1@c2 inPuedeEntrar@rB
link : outHayLugar@rB inHayLugar1@c2

% Acoplamiento salida del c2-rD1
link : outAuto3@c2 inAuto0@rD1
```

```
link : outAuto2@c2 inAuto1@rD1
link : outPuedeSalir3@c2 inPuedeEntrar0@rD1
link : outPuedeSalir2@c2 inPuedeEntrar1@rD1
link : outPuedeEntrar0@rD1 inHayLugar3@c2
link : outPuedeEntrar1@rD1 inHayLugar2@c2


%/******** ACOPLAMIENTOS CRUCE c3 *************************/
% Acoplamiento entrada al cruce rB-c3
link : inAuto@rB inAuto0@c3
link : outHayLugar0@c3 inHayLugar@rB
link : outQuiereCruzar@rB inHayAuto0@c3

% Acoplamiento entrada al cruce rE-c3
link : outAuto@rE inAuto2@c3
link : outHayLugar2@c3 inHayLugar@rE
link : outQuiereCruzar@rE inHayAuto2@c3

% Acoplamiento salida del cruce c3-rC
link : outAuto1@c3 inAuto@rC
link : outPuedeSalir1@c3 inPuedeEntrar@rC
link : outPuedeEntrar@rC inHayLugar1@c3

% ******** ACOPLAMIENTOS CRUCE c4 *************************
% Acoplamiento entrada al cruce rC-c4
link : outAuto@rC inAuto0@c4
link : outHayLugar0@c4 inHayLugar@rC
link : outQuiereCruzar@rC inHayAuto0@c4

% Acoplamiento salida del cruce c4-rF
link : outAuto1@c4 inAuto@rF
link : outPuedeSalir1@c4 inPuedeEntrar@rF
link : outPuedeEntrar@rF inHayLugar1@c4

%*****************************************************************************************
%************************************ Generadores de autos*****************************
%*****************************************************************************************
[GenAuto1]
distribution : constant
value : 0.2
initial : 1
increment : 0

[GenAuto2]
distribution : constant
value : 0.2
initial : 1
increment : 0

[GenAuto3]
distribution : constant
value : 0.5
initial : 1
increment : 0

[GenAuto4]
```

```
distribution : constant
value : 0.3
initial : 1
increment : 0



%*****************************************************************************************
%******************** Transducer  ********************************************************
%*****************************************************************************************
[cont1]
frecuence : 0:0:1:0
TimeUnit : 0:0:1:0

%*****************************************************************************************
%*************************************** Estacion  ***************************************
%*****************************************************************************************
[Estacion]
distribution : constant
value : 1
initial : 1
increment : 0

%*****************************************************************************************
%************************************* Railtrack       ***********************************
%*****************************************************************************************
[via]
type : cell
Width : 4
Height : 1
delay : transport
defaultDelayTime  : #Macro(DemoraVia)
border : nowrapped
neighbors :   via(0,-1)  via(0,0)   via(0,1)
initialvalue : 0
localtransition : pasa-tren
in : inTren inPasaTren0 inPasaTren1 inPasaTren2 inPasaTren3
link : inTren inTren@via(0,0)
link : inPasaTren0 inPasaTren0@via(0,0)
link : inPasaTren1 inPasaTren1@via(0,0)
link : inPasaTren2 inPasaTren2@via(0,0)
link : inPasaTren3 inPasaTren3@via(0,0)
out :  outTren0 outTren1 outPasaTren0 outPasaTren1 outPasaTren2 outPasaTren3
link : outPasaTren0@via(0,0) outPasaTren0
link : outPasaTren1@via(0,0) outPasaTren1
link : outPasaTren2@via(0,0) outPasaTren2
link : outPasaTren3@via(0,0) outPasaTren3
portInTransition : inTren@via(0,0) IngresaTren
portInTransition : inPasaTren0@via(0,0) AvisaPasaTren0
portInTransition : inPasaTren1@via(0,0) AvisaPasaTren1
portInTransition : inPasaTren2@via(0,0) AvisaPasaTren2
portInTransition : inPasaTren3@via(0,0) AvisaPasaTren3


[IngresaTren]
```

```
% Esta ocupada - no ingresa tren desde estacion
rule : { 0 } #Macro(DemoraVia) { (0,0) = 1 }
% Ingresa tren desde estacion
rule : { 1 } #Macro(DemoraVia) { portvalue(thisPort) = 1 }
% Default
rule : { (0,0) } #Macro(DemoraVia) { t }

[pasa-tren]
% Avanza tren hacia adelante
rule : { 0 } #Macro(DemoraVia) { (0,0) = 1 }
% Viene tren de atras
rule : { 1 } #Macro(DemoraVia) { (0,-1) = 1 }
% Default
rule : { (0,0) } #Macro(DemoraVia) { t }

[AvisaPasaTren0]
% Avisa pasa tren
rule : { (0,0) + send(outPasaTren0, 1) } #Macro(DemoraVia) { (0,0)=1 and portvalue(thisPort) = 1 }
% Avisa no pasa tren
rule : { (0,0) + send(outPasaTren0, 0) } #Macro(DemoraVia) { (0,0)=0 and portvalue(thisPort) = 1 }
% Default
rule : { (0,0) } #Macro(DemoraVia) { t }

[AvisaPasaTren1]
% Avisa pasa tren
rule : { (0,0) + send(outPasaTren1, 1) } #Macro(DemoraVia) { (0,0)=1 and portvalue(thisPort) = 1 }
% Avisa no pasa tren
rule : { (0,0) + send(outPasaTren1, 0) } #Macro(DemoraVia) { (0,0)=0 and portvalue(thisPort) = 1 }
% Default
rule : { (0,0) } #Macro(DemoraVia) { t }

[AvisaPasaTren2]
% Avisa pasa tren
rule : { (0,0) + send(outPasaTren2, 1) } #Macro(DemoraVia) { (0,0)=1 and portvalue(thisPort) = 1 }
% Avisa no pasa tren
rule : { (0,0) + send(outPasaTren2, 0) } #Macro(DemoraVia) { (0,0)=0 and portvalue(thisPort) = 1 }
% Default
rule : { (0,0) } #Macro(DemoraVia) { t }

[AvisaPasaTren3]
% Avisa pasa tren
rule : { (0,0) + send(outPasaTren3, 1) } #Macro(DemoraVia) { (0,0)=1 and portvalue(thisPort) = 1 }
% Avisa no pasa tren
rule : { (0,0) + send(outPasaTren3, 0) } #Macro(DemoraVia) { (0,0)=0 and portvalue(thisPort) = 1 }
% Default
rule : { (0,0) } #Macro(DemoraVia) { t }

%*****************************************************************************************
%*********************************  TRAMO rA (1 carril) ********************************
%*****************************************************************************************
[rA]
type : cell
Width : 18
Height : 1
delay : transport
defaultDelayTime  : 10
```

```
border : nowrapped
neighbors : rA(0,-1)  rA(0,0)  rA(0,1)
initialvalue : 0
localtransition : tramo-rule-1carril
in : inAuto inHayLugar
link : inAuto inAuto@rA(0,0)
link : inHayLugar inHayLugar@rA(0,17)
out :  outAuto outQuiereCruzar
link : outAuto@rA(0,17) outAuto
link : outQuiereCruzar@rA(0,17) outQuiereCruzar
portInTransition : inAuto@rA(0,0) IngresaAutoTramo
portInTransition : inHayLugar@rA(0,17) SaleACruceAuto

%*********************************************************************************
%*********************************  TRAMO rB (1 carril) ********************************
%*********************************************************************************
[rB]
type : cell
Width : 10
Height : 1
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rB(0,-1)  rB(0,0)  rB(0,1)
initialvalue : 0
localtransition : tramo-rule-1carril
in : inAuto inPuedeEntrar inhaylugar
link : inAuto inAuto@rB(0,0)
link : inPuedeEntrar inPudeEntrar@rB(0,0)
link : inHayLugar inHayLugar@rB(0,9)
out : outHayLugar outAuto outQuiereCruzar
link : outHayLugar@rB(0,0) outHayLugar
link : outAuto@rB(0,9) outAuto
link : outQuiereCruzar@rB(0,9) outQuiereCruzar
portInTransition : inAuto@rB(0,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rB(0,0) PuedeEntrarTramo
portInTransition : inHayLugar@rB(0,9) SaleACruceAuto

%*********************************************************************************
%*********************************  TRAMO rC (1 carril)  ********************************
%*********************************************************************************
[rC]
type : cell
Width : 14
Height : 1
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rC(0,-1)  rC(0,0)  rC(0,1)
initialvalue : 0
localtransition : tramo-rule-1carril
in : inAuto inPuedeEntrar inHayLugar
link : inAuto inAuto@rC(0,0)
link : inPuedeEntrar inPuedeEntrar@rC(0,0)
link : inHayLugar inHayLugar@rC(0,13)
out :  outPuedeEntrar outAuto outQuiereCruzar
```

```
link : outPuedeEntrar@rC(0,0) outPuedeEntrar
link : outAuto@rC(0,13) outAuto
link : outQuiereCruzar@rC(0,13) outQuiereCruzar
portInTransition : inPuedeEntrar@rC(0,0) PuedeEntrarTramo
portInTransition : inAuto@rC(0,0) IngresaAutoTramo
portInTransition : inHayLugar@rC(0,13) SaleACruceAuto

%*****************************************************************************************
%*********************************  TRAMO rD1 (dos carriles) ***************************
%*****************************************************************************************
[rD1]
type : cell
Width : 17
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rD1(1,-1)   rD1(1,0)   rD1(1,1)
neighbors : rD1(0,-1)   rD1(0,0)   rD1(0,1)
neighbors : rD1(-1,-1)  rD1(-1,0)  rD1(-1,1)
initialvalue : 0
localtransition : tramo-rule-2carril
in : inAuto0 inAuto1 inHayLugar0 inHayLugar1 inPuedeEntrar0 inPuedeEntrar1
link : inAuto0 inAuto@rD1(0,0)
link : inAuto1 inAuto@rD1(1,0)
link : inPuedeEntrar0 inPuedeEntrar@rD1(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rD1(1,0)
link : inHayLugar0 inHayLugar@rD1(0,16)
link : inHayLugar1 inHayLugar@rD1(1,16)
out : outPuedeEntrar0 outPuedeEntrar1 outAuto0 outAuto1 outQuiereCruzar0 outQuiereCruzar1
link : outPuedeEntrar@rD1(0,0) outPuedeEntrar0
link : outPuedeEntrar@rD1(1,0) outPuedeEntrar1
link : outAuto@rD1(0,16) outAuto0
link : outAuto@rD1(1,16) outAuto1
link : outQuiereCruzar@rD1(0,16) outQuiereCruzar0
link : outQuiereCruzar@rD1(1,16) outQuiereCruzar1
portInTransition : inAuto@rD1(0,0) IngresaAutoTramo
portInTransition : inAuto@rD1(1,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rD1(0,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rD1(1,0) PuedeEntrarTramo
portInTransition : inHayLugar@rD1(0,16) SaleACruceAuto
portInTransition : inHayLugar@rD1(1,16) SaleACruceAuto


%*****************************************************************************************
%*********************************  TRAMO rD2 (dos carriles) ***************************
%*****************************************************************************************
[rD2]
type : cell
Width : 17
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rD2(1,-1)   rD2(1,0)  rD2(1,1)
neighbors : rD2(0,-1)   rD2(0,0)  rD2(0,1)
```

```
neighbors : rD2(-1,-1)  rD2(-1,0) rD2(-1,1)
initialvalue : 0
localtransition : tramo-rule-2carril
in : inAuto0 inAuto1 inHayLugar0 inHayLugar1 inPuedeEntrar0 inPuedeEntrar1
link : inAuto0 inAuto@rD2(0,0)
link : inAuto1 inAuto@rD2(1,0)
link : inPuedeEntrar0 inPuedeEntrar@rD2(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rD2(1,0)
link : inHayLugar0 inHayLugar@rD2(0,16)
link : inHayLugar1 inHayLugar@rD2(1,16)
out : outAuto0 outAuto1 outQuiereCruzar0 outQuiereCruzar1 outPuedeEntrar0 outPuedeEntrar1
link : outPuedeEntrar@rD2(0,0) outPuedeEntrar0
link : outPuedeEntrar@rD2(1,0) outPuedeEntrar1
link : outAuto@rD2(0,16) outAuto0
link : outAuto@rD2(1,16) outAuto1
link : outQuiereCruzar@rD2(0,16) outQuiereCruzar0
link : outQuiereCruzar@rD2(1,16) outQuiereCruzar1
portInTransition : inPuedeEntrar@rD2(0,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rD2(1,0) PuedeEntrarTramo
portInTransition : inAuto@rD2(0,0) IngresaAutoTramo
portInTransition : inAuto@rD2(1,0) IngresaAutoTramo
portInTransition : inHayLugar@rD2(0,16) SaleACruceAuto
portInTransition : inHayLugar@rD2(1,16) SaleACruceAuto


%*****************************************************************************************
%*********************************  TRAMO rE (1 carril) ********************************
%*****************************************************************************************
[rE]
type : cell
Width : 14
Height : 1
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rE(0,-1)  rE(0,0)  rE(0,1)
initialvalue : 0
localtransition : tramo-rule-1carril
in : inAuto inPuedeEntrar inHayLugar
link : inAuto inAuto@rE(0,0)
link : inPuedeEntrar inPuedeEntrar@rE(0,0)
link : inHayLugar inHayLugar@rE(0,13)
out : outPuedeEntrar outAuto outQuiereCruzar
link : outPuedeEntrar@rE(0,0) outPuedeEntrar
link : outAuto@rE(0,13) outAuto
link : outQuiereCruzar@rE(0,13) outQuiereCruzar
portInTransition : inAuto@rE(0,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rE(0,0) PuedeEntrarTramo
portInTransition : inHayLugar@rE(0,13) SaleACruceAuto


%*****************************************************************************************
%********************************* TRAMO rF (1 carril) ********************************
%*****************************************************************************************
[rF]
type : cell
```

```
Width : 14
Height : 1
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rF(0,-1)  rF(0,0)  rF(0,1)
initialvalue : 0
localtransition : tramo-rule-1carril
in : inAuto inPuedeEntrar
link : inAuto inAuto@rF(0,0)
link : inPuedeEntrar inPuedeEntrar@rF(0,0)
out : outAuto outPuedeEntrar outQuiereCruzar
link : outPuedeEntrar@rF(0,0) outPuedeEntrar
link : outAuto@rF(0,13) outAuto
link : outQuiereCruzar@rF(0,13) outQuiereCruzar
zone : SaleFueraModelo { (0,13) }
portInTransition : inPuedeEntrar@rF(0,0) PuedeEntrarTramo
portInTransition : inAuto@rF(0,0) IngresaAutoTramo

%*****************************************************************************************
%*********************************  TRAMO rG1 (4 carriles) ***************************
%*****************************************************************************************
[rG1]
type : cell
Width : 27
Height : 4
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rG1(1,-1)  rG1(1,0)  rG1(1,1)
neighbors : rG1(0,-1)  rG1(0,0)  rG1(0,1)
neighbors : rG1(-1,-1) rG1(-1,0) rG1(-1,1)
initialvalue : 0
localtransition : tramo-rule-4carril
in : inAuto0 inAuto1 inAuto2 inAuto3
in : inPuedeEntrar0 inPuedeEntrar1 inPuedeEntrar2 inPuedeEntrar3
in : inHayLugar0 inHayLugar1 inHayLugar2 inHayLugar3
link : inAuto0 inAuto@rG1(0,0)
link : inAuto1 inAuto@rG1(1,0)
link : inAuto2 inAuto@rG1(2,0)
link : inAuto3 inAuto@rG1(3,0)
link : inPuedeEntrar0 inPuedeEntrar@rG1(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rG1(1,0)
link : inPuedeEntrar2 inPuedeEntrar@rG1(2,0)
link : inPuedeEntrar3 inPuedeEntrar@rG1(3,0)
link : inHayLugar0 inHayLugar@rG1(0,26)
link : inHayLugar1 inHayLugar@rG1(1,26)
link : inHayLugar2 inHayLugar@rG1(2,26)
link : inHayLugar3 inHayLugar@rG1(3,26)
out : outPuedeEntrar0 outPuedeEntrar1 outPuedeEntrar2 outPuedeEntrar3
out : outAuto0 outAuto1 outAuto2 outAuto3
out : outQuiereCruzar0 outQuiereCruzar1 outQuiereCruzar2 outQuiereCruzar3
link : outPuedeEntrar@rG1(0,0) outPuedeEntrar0
link : outPuedeEntrar@rG1(1,0) outPuedeEntrar1
link : outPuedeEntrar@rG1(2,0) outPuedeEntrar2
link : outPuedeEntrar@rG1(3,0) outPuedeEntrar3
```

```
link : outAuto@rG1(0,26) outAuto0
link : outAuto@rG1(1,26) outAuto1
link : outAuto@rG1(2,26) outAuto2
link : outAuto@rG1(3,26) outAuto3
link : outQuiereCruzar@rG1(0,26) outQuiereCruzar0
link : outQuiereCruzar@rG1(1,26) outQuiereCruzar1
link : outQuiereCruzar@rG1(2,26) outQuiereCruzar2
link : outQuiereCruzar@rG1(3,26) outQuiereCruzar3
portInTransition : inAuto@rG1(0,0) IngresaAutoTramo
portInTransition : inAuto@rG1(1,0) IngresaAutoTramo
portInTransition : inAuto@rG1(2,0) IngresaAutoTramo
portInTransition : inAuto@rG1(3,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rG1(0,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG1(1,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG1(2,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG1(3,0) PuedeEntrarTramo
portInTransition : inHayLugar@rG1(0,26) SaleACruceAuto
portInTransition : inHayLugar@rG1(1,26) SaleACruceAuto
portInTransition : inHayLugar@rG1(2,26) SaleACruceAuto
portInTransition : inHayLugar@rG1(3,26) SaleACruceAuto

%*****************************************************************************************
%****************************** *** TRAMO rG2 (4 carriles) ***************************
%*****************************************************************************************
[rG2]
type : cell
Width : 27
Height : 4
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rG2(1,-1)  rG2(1,0)  rG2(1,1)
neighbors : rG2(0,-1)  rG2(0,0)  rG2(0,1)
neighbors : rG2(-1,-1) rG2(-1,0) rG2(-1,1)
initialvalue : 0
localtransition : tramo-rule-4carril
in : inAuto0 inAuto1 inAuto2 inAuto3
in : inPuedeEntrar0 inPuedeEntrar1 inPuedeEntrar2 inPuedeEntrar3
in : inHayLugar0 inHayLugar1 inHayLugar2 inHayLugar3
link : inAuto0 inAuto@rG2(0,0)
link : inAuto1 inAuto@rG2(1,0)
link : inAuto2 inAuto@rG2(2,0)
link : inAuto3 inAuto@rG2(3,0)
link : inPuedeEntrar0 inPuedeEntrar@rG2(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rG2(1,0)
link : inPuedeEntrar2 inPuedeEntrar@rG2(2,0)
link : inPuedeEntrar3 inPuedeEntrar@rG2(3,0)
link : inHayLugar0 inHayLugar@rG2(0,26)
link : inHayLugar1 inHayLugar@rG2(1,26)
link : inHayLugar2 inHayLugar@rG2(2,26)
link : inHayLugar3 inHayLugar@rG2(3,26)
out : outPuedeEntrar0 outPuedeEntrar1 outPuedeEntrar2 outPuedeEntrar3
out : outAuto0 outAuto1 outAuto2 outAuto3
out : outQuiereCruzar0 outQuiereCruzar1  outQuiereCruzar2 outQuiereCruzar3
link : outPuedeEntrar@rG2(0,0) outPuedeEntrar0
link : outPuedeEntrar@rG2(1,0) outPuedeEntrar1
```

```
link : outPuedeEntrar@rG2(2,0) outPuedeEntrar2
link : outPuedeEntrar@rG2(3,0) outPuedeEntrar3
link : outAuto@rG2(0,26) outAuto0
link : outAuto@rG2(1,26) outAuto1
link : outAuto@rG2(2,26) outAuto2
link : outAuto@rG2(3,26) outAuto3
link : outQuiereCruzar@rG2(0,26) outQuiereCruzar0
link : outQuiereCruzar@rG2(1,26) outQuiereCruzar1
link : outQuiereCruzar@rG2(2,26) outQuiereCruzar2
link : outQuiereCruzar@rG2(3,26) outQuiereCruzar3
zone : SaleFueraModelo { (0,26), (1,26), (2,26), (3,26) }
portInTransition : inAuto@rG2(0,0) IngresaAutoTramo
portInTransition : inAuto@rG2(1,0) IngresaAutoTramo
portInTransition : inAuto@rG2(2,0) IngresaAutoTramo
portInTransition : inAuto@rG2(3,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rG2(0,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG2(1,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG2(2,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rG2(3,0) PuedeEntrarTramo
portInTransition : inHayLugar@rG2(0,26) SaleACruceAuto
portInTransition : inHayLugar@rG2(1,26) SaleACruceAuto
portInTransition : inHayLugar@rG2(2,26) SaleACruceAuto
portInTransition : inHayLugar@rG2(3,26) SaleACruceAuto

%*****************************************************************************************
%********************************** TRAMO rH1 (2 carriles) ******************************
%*****************************************************************************************
[rH1]
type : cell
Width : 14
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rH1(1,-1)  rH1(1,0)  rH1(1,1)
neighbors : rH1(0,-1)  rH1(0,0)  rH1(0,1)
neighbors : rH1(-1,-1) rH1(-1,0) rH1(-1,1)
initialvalue : 0
localtransition : tramo-rule-2carril
in : inAuto0 inAuto1 inPuedeEntrar0 inPuedeEntrar1
link : inAuto0 inAuto@rH1(0,0)
link : inAuto1 inAuto@rH1(1,0)
link : inPuedeEntrar0 inPuedeEntrar@rH1(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rH1(1,0)
out : outAuto0 outAuto1 outQuiereCruzar0 outQuiereCruzar1 outPuedeEntrar0 outPuedeEntrar1
link : outPuedeEntrar@rH1(0,0) outPuedeEntrar0
link : outPuedeEntrar@rH1(1,0) outPuedeEntrar1
link : outAuto@rH1(0,13) outAuto0
link : outAuto@rH1(1,13) outAuto1
link : outQuiereCruzar@rH1(0,13) outQuiereCruzar0
link : outQuiereCruzar@rH1(1,13) outQuiereCruzar1
zone : SaleFueraModelo { (0,13), (1,13) }
portInTransition : inPuedeEntrar@rH1(0,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rH1(1,0) PuedeEntrarTramo
portInTransition : inAuto@rH1(0,0) IngresaAutoTramo
portInTransition : inAuto@rH1(1,0) IngresaAutoTramo
```

```
%*****************************************************************************************
%*********************************  TRAMO rH2 (2 carriles) ******************************
%*****************************************************************************************
[rH2]
type : cell
Width : 14
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rH2(1,-1)  rH2(1,0)  rH2(1,1)
neighbors : rH2(0,-1)  rH2(0,0)  rH2(0,1)
neighbors : rH2(-1,-1) rH2(-1,0) rH2(-1,1)
initialvalue : 0
localtransition : tramo-rule-2carril
in : inAuto0 inAuto1 inHayLugar0 inHayLugar1
link : inAuto0 inAuto@rH2(0,0)
link : inAuto1 inAuto@rH2(1,0)
link : inHayLugar0 inHayLugar@rH2(0,13)
link : inHayLugar1 inHayLugar@rH2(1,13)
out : outAuto0 outAuto1 outQuiereCruzar0 outQuiereCruzar1
link : outAuto@rH2(0,13) outAuto0
link : outAuto@rH2(1,13) outAuto1
link : outQuiereCruzar@rH2(0,13) outQuiereCruzar0
link : outQuiereCruzar@rH2(1,13) outQuiereCruzar1
portInTransition : inAuto@rH2(0,0) IngresaAutoTramo
portInTransition : inAuto@rH2(1,0) IngresaAutoTramo
portInTransition : inHayLugar@rH2(0,13) SaleACruceAuto
portInTransition : inHayLugar@rH2(1,13) SaleACruceAuto


%*****************************************************************************************
%*********************************  TRAMO rI1 (2 carriles) ******************************
%********************************* Por este tramo pasa un tren  ****************************
%*****************************************************************************************
[rI1]
type : cell
Width : 18
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rI1(1,-1)  rI1(1,0)  rI1(1,1)
neighbors : rI1(0,-1)  rI1(0,0)  rI1(0,1)
neighbors : rI1(-1,-1) rI1(-1,0) rI1(-1,1)
initialvalue : 0
localtransition : tramo-rule-2carril
in : inAuto0 inAuto1 inPuedeEntrar0 inPuedeEntrar1 inPasaTrenAntes0 inPasaTrenAntes1
in : inPasaTrenDespues0 inPasaTrenDespues1
link : inAuto0 inAuto@rI1(0,0)
link : inAuto1 inAuto@rI1(1,0)
link : inPuedeEntrar0 inPuedeEntrar@rI1(0,0)
link : inPuedeEntrar1 inPuedeEntrar@rI1(1,0)
link : inPasaTrenAntes0 inPasaTren@rI1(0,12)
link : inPasaTrenAntes1 inPasaTren@rI1(1,12)
```

```
link : inPasaTrenDespues0 inPasaTren@rI1(0,13)
link : inPasaTrenDespues1 inPasaTren@rI1(1,13)
out : outPuedeEntrar0 outPuedeEntrar1  outAuto0 outAuto1
out : outPasaTren0 outPasaTren1 outQuiereCruzar0 outQuiereCruzar1
link : outPuedeEntrar@rI1(0,0) outPuedeEntrar0
link : outPuedeEntrar@rI1(1,0) outPuedeEntrar1
link : outAuto@rI1(0,17) outAuto0
link : outAuto@rI1(1,17) outAuto1
link : outPasaTren@rI1(0,12) outPasaTren0
link : outPasaTren@rI1(1,12) outPasaTren1
link : outQuiereCruzar@rI1(0,17) outQuiereCruzar0
link : outQuiereCruzar@rI1(1,17) outQuiereCruzar1
zone : AntesTren { (0,12), (1,12) }
zone : DespuesTren { (0,13), (1,13) }
zone : SaleFueraModelo { (0,17), (1,17) }
portInTransition : inPasaTren@rI1(0,12) AvisoPasoTren
portInTransition : inPasaTren@rI1(1,12) AvisoPasoTren
portInTransition : inPasaTren@rI1(0,13) AvisoPasoTren
portInTransition : inPasaTren@rI1(1,13) AvisoPasoTren
portInTransition : inAuto@rI1(0,0) IngresaAutoTramo
portInTransition : inAuto@rI1(1,0) IngresaAutoTramo
portInTransition : inPuedeEntrar@rI1(0,0) PuedeEntrarTramo
portInTransition : inPuedeEntrar@rI1(1,0) PuedeEntrarTramo

%*********************************************************************************************
%*********************************  TRAMO rI2 (2 carriles) ***************************
%*********************************Por este tramo pasa un tren  ************************
%*********************************************************************************************
[rI2]
type : cell
Width : 18
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : rI2(1,-1)  rI2(1,0)  rI2(1,1)
neighbors : rI2(0,-1)  rI2(0,0)  rI2(0,1)
neighbors : rI2(-1,-1) rI2(-1,0) rI2(-1,1)
initialvalue : 0
localtransition : tramo-rule-2carril
in : inAuto0 inAuto1 inHayLugar0 inHayLugar1
in : inPasaTrenAntes0 inPasaTrenAntes1 inPasaTrenDespues0 inPasaTrenDespues1
link : inAuto0 inAuto@rI2(0,0)
link : inAuto1 inAuto@rI2(1,0)
link : inHayLugar0 inHayLugar@rI2(0,17)
link : inHayLugar1 inHayLugar@rI2(1,17)
link : inPasaTrenAntes0 inPasaTren@rI2(0,3)
link : inPasaTrenAntes1 inPasaTren@rI2(1,3)
link : inPasaTrenDespues0 inPasaTren@rI2(0,4)
link : inPasaTrenDespues1 inPasaTren@rI2(1,4)
out : outAuto0 outAuto1 outQuiereCruzar0 outQuiereCruzar1
out : outPasaTren0 outPasaTren1
link : outAuto@rI2(0,17) outAuto0
link : outAuto@rI2(1,17) outAuto1
link : outQuiereCruzar@rI2(0,17) outQuiereCruzar0
link : outQuiereCruzar@rI2(1,17) outQuiereCruzar1
```

```
link : outPasaTren@rI2(0,3) outPasaTren0
link : outPasaTren@rI2(1,3) outPasaTren1
zone : AntesTren { (0,3), (1,3) }
zone : DespuesTren { (0,4) ,(1,4) }
portInTransition : inAuto@rI2(0,0) IngresaAutoTramo
portInTransition : inAuto@rI2(1,0) IngresaAutoTramo
portInTransition : inHayLugar@rI2(0,17) SaleACruceAuto
portInTransition : inHayLugar@rI2(1,17) SaleACruceAuto
portInTransition : inPasaTren@rI2(0,3) AvisoPasoTren
portInTransition : inPasaTren@rI2(1,3) AvisoPasoTren
portInTransition : inPasaTren@rI2(0,4) AvisoPasoTren
portInTransition : inPasaTren@rI2(1,4) AvisoPasoTren


%********** Reglas que tienen que ver con el paso del tren *********************************
[AntesTren]
% Antes de avanzar debe preguntar si esta pasando un tren
rule : { (0,0) + send(outPasaTren, 1) } 10 { (0,0) = 1 and (0,1) = 0 }
% Viene de atras
rule : { 1 } 10 { (0,-1) =1  and (0,0) = 0 }
% Default
rule : {(0,0)} 10  { t }

[DespuesTren]
% Avanza hacia adelante
rule : { 0 } 10 { (0,0) = 1 and (0,1) = 0 }
% Default
rule : {(0,0)} 10  { t }


[AvisoPasoTren]
% Avanza si pasa el tren
rule : { 0 } 10 { (0,0) = 1 and (0,1) = 0 and portvalue(ThisPort) = 0 }
% Avanza si pasa el tren
rule : { 1 } 10 { (0,0) = 0 and (0,-1) = 1 and portvalue(ThisPort) = 0 }
% Default
rule : {(0,0)} 10  { t }

%*****************************************************************************************
%************************** REGLAS COMUNES A LOS TRAMOS ************************
%*****************************************************************************************
[PuedeEntrarTramo]
% Le indica al cruce si hay lugar para que ingrese el auto
rule : { (0,0) + send(outPuedeEntrar,(0,0)) } 10 { t }

[IngresaAutoTramo]
% Ingresa un auto al tramo (posiblemente desde un cruce)
rule : { 1 } 10 { portvalue(ThisPort)!= 0 and (0,0) = 0 }
rule : 0 10  { t }

[tramo-rule-1carril]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) = 1 and (0,1) = 0 }
% Viene de atras
rule : { 1 } 10 { not isUndefined((0,-1)) and (0,-1) = 1  and (0,0) = 0 }
% Quiere salir a cruce
```

```
rule : {(0,0) + send(outQuiereCruzar, 1)} 10 { not isundefined((0,-1)) and isundefined((0,1)) and  (0,0) = 1}
% Default
rule : {(0,0)} 10  { t }


[tramo-rule-2carril]
%  Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) = 1 and (0,1) = 0 }
% Viene de atras
rule : { 1 } 10 { not isUndefined((0,-1)) and (0,-1) =1  and (0,0) = 0 }
% Quiere salir a cruce
rule : {(0,0) + send(outQuiereCruzar, 1)} 10 { not isundefined((0,-1)) and isundefined((0,1)) and  (0,0) = 1}
% Carril 0 - Viene de atras en diagonal del Carril 1 al Carril 0
rule : {1} 10 { not isUndefined((-1,0)) and not isUndefined((-1,-1)) and not isUndefined((0,-1)) and (0,0) = 0 and
(0, -1)= 0 and (-1,-1) =1 and (-1,0) = 1}
% Carril 0 - Sale hacia adelante en diagonal del Carril 0 al Carril 1
rule : {0} 10 { not isundefined ((-1,0)) and not isundefined((0,1)) and (0,0) = 1 and (-1,1)=0 and (-1,0) =0 }
% Carril 1 - Viene de atras en diagonal del Carril 0 al Carril 1
rule : {1} 10 { isundefined((-1,0)) and not isundefined ((1,0)) and (0,0) = 0 and (0, -1)=0 and (1,-1) =1 and (1,0) =
1}
% Carril 1 - Sale hacia adelante en diagonal del Carril 1 al Carril 0
rule : {0} 10 { isundefined((-1,0)) and not isundefined ((1,0)) and not isundefined((0,1)) and (0,0) = 1 and (1,1)=0
and (-1,0) =0 and (1,0) = 0}
% Default
rule : {(0,0)} 10  { t }

[tramo-rule-4carril]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) = 1 and (0,1) = 0 }
% Viene de atras
rule : { 1 } 10 { not isUndefined((0,-1)) and (0,-1) =1  and (0,0) = 0 }
% Quiere salir a cruce
rule : {(0,0) + send(outQuiereCruzar, 1)} 10 { not isundefined((0,-1)) and isundefined((0,1)) and  (0,0) = 1}
% Default
rule : {(0,0)} 10  { t }

[SaleACruceAuto]
% Sale hacia adelante (para la celda de salida)
rule : { 0 + send(outAuto,(0,0) ) } 0 {(0,0)=1  and portvalue(ThisPort) = 0 }
rule : {(0,0)} 10  { t }

[SaleFueraModelo]
% Viene de delante
rule : {1} 10 {(0,0) = 0 and (0,-1) =1 }
% Sale fuera del modelo
rule : {0 + send(outAuto,1)} 10 {(0,0) = 1 }
% Default
rule : {(0,0)} 10  { t }


%*****************************************************************************
%****************************** CRUCE c2 ***********************************
%*****************************************************************************
[c2]
type : cell
Width : 6
```

```
Height : 1
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : c2(0,-1) c2(0,0) c2(0,1)
initialvalue : 0
localtransition : autoCruce-entrada
in : inAuto0 inHayAuto0 inHayLugar1 inHayLugar2 inHayLugar3 inAuto4 inHayAuto4
in : inAuto5 inHayAuto5
link : inAuto0 inAuto@c2(0,0)
link : inHayAuto0 inHayAuto@c2(0,0)
link : inHayLugar1 inHayLugar@c2(0,1)
link : inHayLugar2 inHayLugar@c2(0,2)
link : inHayLugar3 inHayLugar@c2(0,3)
link : inAuto4 inAuto@c2(0,4)
link : inHayAuto4 inHayAuto@c2(0,4)
link : inAuto5 inAuto@c2(0,5)
link : inHayAuto5 inHayAuto@c2(0,5)
out : outHayLugar0 outAuto1 outPuedeSalir1 outAuto2 outPuedeSalir2
out : outAuto3 outPuedeSalir3 outHayLugar4 outHayLugar5
link : outHayLugar@c2(0,0) outHayLugar0
link : outAuto@c2(0,1) outAuto1
link : outPuedeSalir@c2(0,1) outPuedeSalir1
link : outAuto@c2(0,2) outAuto2
link : outPuedeSalir@c2(0,2) outPuedeSalir2
link : outAuto@c2(0,3) outAuto3
link : outPuedeSalir@c2(0,3) outPuedeSalir3
link : outHayLugar@c2(0,4) outHayLugar4
link : outHayLugar@c2(0,5) outHayLugar5
zone : AutoCruce-salida { (0,1), (0,2), (0,3) }
portInTransition : inAuto@c2(0,0) IngresaAutoCruce
portInTransition : inHayAuto@c2(0,0) QuiereCruzar
portInTransition : inHayLugar@c2(0,1) PuedeSalirAutoCruce
portInTransition : inHayLugar@c2(0,2) PuedeSalirAutoCruce
portInTransition : inHayLugar@c2(0,3) PuedeSalirAutoCruce
portInTransition : inAuto@c2(0,4) IngresaAutoCruce
portInTransition : inHayAuto@c2(0,4) QuiereCruzar
portInTransition : inAuto@c2(0,5) IngresaAutoCruce
portInTransition : inHayAuto@c2(0,5) QuiereCruzar

%*****************************************************************************************
%******************************** CRUCE c3 ***********************************************
%*****************************************************************************************
[c3]
type : cell
Width : 3
Height : 1
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : c3(0,-1) c3(0,0) c3(0,1)
initialvalue : 0
localtransition : autoCruce-entrada
in : inAuto0 inHayAuto0 inHayLugar1 inAuto2 inHayAuto2
link : inAuto0 inAuto@c3(0,0)
link : inHayAuto0 inHayAuto@c3(0,0)
```

```
link : inHayLugar1 inHayLugar@c3(0,1)
link : inAuto2 inAuto@c3(0,2)
link : inHayAuto2 inHayAuto@c3(0,2)
out : outHayLugar0 outAuto1 outPuedeSalir1 outHayLugar2
link : outHayLugar@c3(0,0) outHayLugar0
link : outAuto@c3(0,1) outAuto1
link : outPuedeSalir@c3(0,1) outPuedeSalir1
link : outHayLugar@c3(0,2) outHayLugar2
zone : AutoCruce-salida { (0,1) }
portInTransition : inAuto@c3(0,0) IngresaAutoCruce
portInTransition : inHayAuto@c3(0,0) QuiereCruzar
portInTransition : inHayLugar@c3(0,1) PuedeSalirAutoCruce
portInTransition : inAuto@c3(0,2) IngresaAutoCruce
portInTransition : inHayAuto@c3(0,2) QuiereCruzar

%*****************************************************************************************
%******************************** CRUCE c4 ***********************************************
%*****************************************************************************************
[c4]
type : cell
Width : 2
Height : 1
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : c4(0,-1) c4(0,0) c4(0,1)
initialvalue : 0
localtransition : autoCruce-entrada
in : inAuto0 inHayAuto0 inHayLugar1
link : inAuto0 inAuto@c4(0,0)
link : inHayAuto0 inHayAuto@c4(0,0)
link : inHayLugar1 inHayLugar@c4(0,1)
out : outAuto1 outHayLugar0 outPuedeSalir1
link : outHayLugar@c4(0,0) outHayLugar0
link : outAuto@c4(0,1) outAuto1
link : outPuedeSalir@c4(0,1) outPuedeSalir1
zone : AutoCruce-salida { (0,1) }
portInTransition : inAuto@c4(0,0) IngresaAutoCruce
portInTransition : inHayAuto@c4(0,0) QuiereCruzar
portInTransition : inHayLugar@c4(0,1) PuedeSalirAutoCruce

%*****************************************************************************************
%******************************** CRUCE c1 ***********************************************
%*****************************************************************************************
[c1]
type : cell
Width : 21
Height : 1
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : c1(0,-1) c1(0,0) c1(0,1)
initialvalue : 0
localtransition : autoCruce-entrada
in : inHayLugar0 inHayLugar1 inHayLugar2 inHayLugar3 inAuto4 inHayAuto4 inAuto5 inHayAuto5
in : inAuto6 inHayAuto6 inAuto7 inHayAuto7 inHayLugar8 inHayLugar9
```

```
in : inAuto10 inHayAuto10 inAuto11 inHayAuto11 inHayLugar12 inHayLugar13 inHayLugar14
in : inAuto15 inHayAuto15 inAuto16 inHayAuto16 inHayLugar17 inHayLugar18
in : inAuto19 inHayAuto19 inAuto20 inHayAuto20
link : inHayLugar0 inHayLugar@c1(0,0)
link : inHayLugar1 inHayLugar@c1(0,1)
link : inHayLugar2 inHayLugar@c1(0,2)
link : inHayLugar3 inHayLugar@c1(0,3)
link : inAuto4 inAuto@c1(0,4)
link : inHayAuto4 inHayAuto@c1(0,4)
link : inAuto5 inAuto@c1(0,5)
link : inHayAuto5 inHayAuto@c1(0,5)
link : inAuto6 inAuto@c1(0,6)
link : inHayAuto6 inHayAuto@c1(0,6)
link : inAuto7 inAuto@c1(0,7)
link : inHayAuto7 inHayAuto@c1(0,7)
link : inHayLugar8 inHayLugar@c1(0,8)
link : inHayLugar9 inHayLugar@c1(0,9)
link : inAuto10 inAuto@c1(0,10)
link : inHayAuto10 inHayAuto@c1(0,10)
link : inAuto11 inAuto@c1(0,11)
link : inHayAuto11 inHayAuto@c1(0,11)
link : inHayLugar12 inHayLugar@c1(0,12)
link : inHayLugar13 inHayLugar@c1(0,13)
link : inHayLugar14 inHayLugar@c1(0,14)
link : inAuto15 inAuto@c1(0,15)
link : inHayAuto15 inHayAuto@c1(0,15)
link : inAuto16 inAuto@c1(0,16)
link : inHayAuto16 inHayAuto@c1(0,16)
link : inHayLugar17 inHayLugar@c1(0,17)
link : inHayLugar18 inHayLugar@c1(0,18)
link : inAuto19 inAuto@c1(0,19)
link : inHayAuto19 inHayAuto@c1(0,19)
link : inAuto20 inAuto@c1(0,20)
link : inHayAuto20 inHayAuto@c1(0,20)
out : outAuto0 outPuedeSalir0 outAuto1 outPuedeSalir1 outAuto2 outPuedeSalir2
out : outAuto3 outPuedeSalir3 outHayLugar4 outHayLugar5 outHayLugar6 outHayLugar7
out : outAuto8 outPuedeSalir8  outAuto9 outPuedeSalir9 outHayLugar10 outHayLugar11
out : outAuto12 outPuedeSalir12 outAuto13 outPuedeSalir13 outAuto14 outPuedeSalir14
out : outHayLugar15 outHayLugar16 outAuto17 outPuedeSalir17 outAuto18 outPuedeSalir18
out : outHayLugar19 outHayLugar20
link : outAuto@c1(0,0) outAuto0
link : outPuedeSalir@c1(0,0) outPuedeSalir0
link : outAuto@c1(0,1) outAuto1
link : outPuedeSalir@c1(0,1) outPuedeSalir1
link : outAuto@c1(0,2) outAuto2
link : outPuedeSalir@c1(0,2) outPuedeSalir2
link : outAuto@c1(0,3) outAuto3
link : outPuedeSalir@c1(0,3) outPuedeSalir3
link : outHayLugar@c1(0,4) outHayLugar4
link : outHayLugar@c1(0,5) outHayLugar5
link : outHayLugar@c1(0,6) outHayLugar6
link : outHayLugar@c1(0,7) outHayLugar7
link : outAuto@c1(0,8) outAuto8
link : outPuedeSalir@c1(0,8) outPuedeSalir8
link : outAuto@c1(0,9) outAuto9
link : outPuedeSalir@c1(0,9) outPuedeSalir9
```

```
link : outHayLugar@c1(0,10) outHayLugar10
link : outHayLugar@c1(0,11) outHayLugar11
link : outAuto@c1(0,12) outAuto12
link : outPuedeSalir@c1(0,12) outPuedeSalir12
link : outAuto@c1(0,13) outAuto13
link : outPuedeSalir@c1(0,13) outPuedeSalir13
link : outAuto@c1(0,14) outAuto14
link : outPuedeSalir@c1(0,14) outPuedeSalir14
link : outHayLugar@c1(0,15) outHayLugar15
link : outHayLugar@c1(0,16) outHayLugar16
link : outAuto@c1(0,17) outAuto17
link : outPuedeSalir@c1(0,17) outPuedeSalir17
link : outAuto@c1(0,18) outAuto18
link : outPuedeSalir@c1(0,18) outPuedeSalir18
link : outHayLugar@c1(0,19) outHayLugar19
link : outHayLugar@c1(0,20) outHayLugar20
zone : AutoCruce-salida { (0,0) ,(0,1), (0,2), (0,3), (0,8), (0,9), (0,12), (0,13), (0,14), (0,17), (0,18) }
%zone : ZonaBache { (0,4), (0,5), (0,6), (0,7) }
portInTransition : inHayLugar@c1(0,0) PuedeSalirAutoCruce
portInTransition : inHayLugar@c1(0,1) PuedeSalirAutoCruce
portInTransition : inHayLugar@c1(0,2) PuedeSalirAutoCruce
portInTransition : inHayLugar@c1(0,3) PuedeSalirAutoCruce
portInTransition : inAuto@c1(0,4) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,4) QuiereCruzar
portInTransition : inAuto@c1(0,5) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,5) QuiereCruzar
portInTransition : inAuto@c1(0,6) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,6) QuiereCruzar
portInTransition : inAuto@c1(0,7) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,7) QuiereCruzar
portInTransition : inHayLugar@c1(0,8) PuedeSalirAutoCruce
portInTransition : inAuto@c1(0,9) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,9) QuiereCruzar
portInTransition : inAuto@c1(0,10) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,10) QuiereCruzar
portInTransition : inHayLugar@c1(0,12) PuedeSalirAutoCruce
portInTransition : inHayLugar@c1(0,13) PuedeSalirAutoCruce
portInTransition : inHayLugar@c1(0,14) PuedeSalirAutoCruce
portInTransition : inAuto@c1(0,15) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,15) QuiereCruzar
portInTransition : inAuto@c1(0,16) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,16) QuiereCruzar
portInTransition : inHayLugar@c1(0,17) PuedeSalirAutoCruce
portInTransition : inHayLugar@c1(0,18) PuedeSalirAutoCruce
portInTransition : inAuto@c1(0,19) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,19) QuiereCruzar
portInTransition : inAuto@c1(0,20) IngresaAutoCruce
portInTransition : inHayAuto@c1(0,20) QuiereCruzar

%*************************************************************************************
%************************* REGLAS COMUNES A LOS CRUCES ************************
%*************************************************************************************
[AutoCruce-entrada]
% Ingresa a la celda un auto que ya estaba dentro del cruce
rule : 1 10 { (0,0) = 0 and (0,-1) =1}
% Avanza a la prox celda
```

```
rule : 0 10 { (0,0) = 1 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[ZonaBache]
% Ingresa a la celda un auto que ya estaba dentro del cruce
rule : 1 800 { (0,0) = 0 and (0,-1) =1}
% Avanza a la prox celda
rule : 0 800 { (0,0) = 1 and (0,1) = 0 }
rule : {(0,0)} 800  { t }

[AutoCruce-salida]
% Avanza el auto si no debe salir por el tramo (de acuerdo al camino)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1)= 1 and not #Macro(Salir) }
% Le indica al tramo que hay un auto que puede entrar al mismo (si corresponde al camino)
rule : { 0 + send(outPuedeSalir,1)} 10 { (0,0) = 0 and (0,-1)=1  and #Macro(Salir)}
% Avance de un auto dentro del cruce
rule : 0 10 { (0,0)=1 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[IngresaAutoCruce]
rule : {portvalue(ThisPort)} 10 { (0,0) = 0 and portvalue(ThisPort) = 1}
rule : {(0,0)} 10 { t }

[IngresaAutoCruceBache]
rule : {portvalue(ThisPort)} 800 { (0,0) = 0 and portvalue(ThisPort) = 1}
rule : {(0,0)} 800 { t }

[PuedeSalirAutoCruce]
% Llega auto que abandonara el cruce
rule : {0 + send(outAuto, (0,-1)) } 10 { (0,0) = 0 and (0,-1)=1 and portvalue(ThisPort) = 0 and #Macro(Salir)}
% Llega auto que permanecera dentro del cruce
%rule : {0 + send(outAuto, 0)} 10 { (0,0) = 0 and  (0,-1)=1 and ( portvalue(ThisPort) = 1 or ( portvalue(ThisPort) =
0 and not #Macro(Salir) }
rule : {0 + send(outAuto, 0)} 10 { (0,0) = 0 and  (0,-1)=1 and  portvalue(ThisPort) = 1 }
rule : {(0,0)} 10  { t }

[QuiereCruzar]
rule : { (0,0) + send(outhaylugar,0) } 10 { (0,0) = 0 and (0,-1) = 0 and portvalue(ThisPort) = 1 }
rule : { (0,0) + send(outhayLugar,1) } 10  { ((0,0) = 1 or ((0,0) = 0 and (0,-1) = 1)) and portvalue(ThisPort) = 1 }
rule : {(0,0)} 10  { t }
```

1.2.2. **Archivo : ciudad.inc**

```
#BeginMacro(Salir)
 (random >= 0.3 )
#EndMacro

#BeginMacro(DemoraVia)
  500
#EndMacro
```

1.3. Ejemplo de ruteo dinámico

1.3.1. Archivo : rd.ma

```
#include(rd.inc)

[top]
components : tramo1 tramo2 tramo3 tramo4 tramo5 cruce1 cruce2 cruce3 cruce4
components : congest2@congestion congest3@congestion od1@od

in : inNuevoAuto inNuevoCamino
%* Puertos de entrada al modelo acoplado
link : inNuevoAuto inAuto@tramo1
link : inNuevoCamino inCamino@tramo1

% Acoplamiento Tramo1-Cruce1
link : outAuto@Tramo1 inAuto@Cruce1
link : outCamino@Tramo1 inCamino@Cruce1
link : outQuiereCruzar@Tramo1 inHayAuto@Cruce1
link : outHayLugarParaEntrar@Cruce1 inHayLugar@Tramo1

% Acoplamiento Cruce1-Tramo2
link : outAuto2@Cruce1 inAuto@Tramo2
link : outCamino2@Cruce1 inCamino@Tramo2
link : outPuedeSalir2@Cruce1 inPuedeEntrar@Tramo2
link : outPuedeEntrar@Tramo2 inHayLugar2@Cruce1

% Acoplamiento Cruce1-Tramo3
link : outAuto3@Cruce1 inAuto@Tramo3
link : outCamino3@Cruce1 inCamino@Tramo3
link : outPuedeSalir3@Cruce1 inPuedeEntrar@Tramo3
link : outPuedeEntrar@Tramo3 inHayLugar3@Cruce1

% Acoplamiento Cruce1-Congest2
link : outSolicitaPeso2@Cruce1 PreguntaPeso@congest2
link : peso@congest2 inPeso@Cruce1
link : peso@congest2 inPesoCamino@Cruce1
link : peso@congest2 inPesoRuteado@Cruce1

% Acoplamiento Cruce1-Congest3
link : outSolicitaPeso3@Cruce1 PreguntaPeso@congest3
link : peso@congest3 inPeso@Cruce1
link : peso@congest3 inPesoCamino@Cruce1
link : peso@congest3 inPesoRuteado@Cruce1

% Acoplamiento Cruce1-OD1
link : outSolicitaCamino@cruce1 SolicitaCamino@od1
link : InformaCamino@od1 inNuevoCamino@cruce1

% Acoplamiento Tramo2-Congest2
link : outEntraAuto@tramo2 IngresaAuto@congest2
link : outAuto@tramo2 SaleAuto@congest2

% Acoplamiento Tramo2-Congest3
link : outEntraAuto@tramo3 IngresaAuto@congest3
```

```
link : outAuto@tramo3 SaleAuto@congest3

% Acoplamiento Tramo2-Cruce2
link : outAuto@Tramo2 inAuto@Cruce2
link : outCamino@Tramo2 inCamino@Cruce2
link : outHayLugarParaEntrar@Cruce2 inHayLugar@Tramo2
link : outQuiereCruzar@Tramo2 inHayAuto@Cruce2

% Acoplamiento Tramo3-Cruce3
link : outAuto@Tramo3 inAuto@Cruce3
link : outCamino@Tramo3 inCamino@Cruce3
link : outHayLugarParaEntrar@Cruce3 inHayLugar@Tramo3
link : outQuiereCruzar@Tramo3 inHayAuto@Cruce3

% Acoplamiento Cruce2-Tramo5
link : outAuto@Cruce2 inAuto@Tramo5
link : outCamino@Cruce2 inCamino@Tramo5
link : outPuedeSalir5@Cruce2 inPuedeEntrar@Tramo5
link : outPuedeEntrar@Tramo5 inHayLugar@Cruce2

% Acoplamiento Cruce3-Tramo4
link : outAuto@Cruce3 inAuto@Tramo4
link : outCamino@Cruce3 inCamino@Tramo4
link : outPuedeSalir4@Cruce3 inPuedeEntrar@Tramo4
link : outPuedeEntrar@Tramo4 inHayLugar@Cruce3

% Acoplamiento Tramo4-Cruce4
link : outAuto@Tramo4 inAuto4@Cruce4
link : outCamino@Tramo4 inCamino4@Cruce4
link : outHayLugarParaEntrar4@Cruce4 inHayLugar@Tramo4
link : outQuiereCruzar@Tramo4 inHayAuto4@Cruce4

% Acoplamiento Tramo5-Cruce4
link : outAuto@Tramo5 inAuto5@Cruce4
link : outCamino@Tramo5 inCamino5@Cruce4
link : outHayLugarParaEntrar5@Cruce4 inHayLugar@Tramo5
link : outQuiereCruzar@Tramo5 inHayAuto5@Cruce4


%*******************************************************************************
%*******************************  TRAMO 1 **************************************
%*******************************************************************************
[tramo1]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo1(0,-1)  tramo1(0,0)  tramo1(0,1)
neighbors : tramo1(1,-1)  tramo1(1,0)  tramo1(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar
link : inAuto inAuto@Tramo1(0,0)
link : inCamino inCamino@Tramo1(1,0)
```

```
link : inHayLugar inHayLugar@Tramo1(0,4)
link : inHayLugar inHayLugar@Tramo1(1,4)
out : outQuiereCruzar outAuto outCamino
link : outQuiereCruzar@tramo1(0,4) outQuiereCruzar
link : outAuto@tramo1(0,4) outAuto
link : outCamino@tramo1(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inAuto@tramo1(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo1(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo1(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo1(1,4) SaleACruceCamino

%*****************************************************************************************
%****************************** Modelos de Congestion ****************************
%*****************************************************************************************
[congest2]
autos: 0

[congest3]
autos: 0

%*****************************************************************************************
%****************************** TRAMO 2 ****************************************
%*****************************************************************************************
[tramo2]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo2(0,-1)  tramo2(0,0)  tramo2(0,1)
neighbors : tramo2(1,-1)  tramo2(1,0)  tramo2(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
link : inPuedeEntrar inPuedeEntrar@Tramo2(0,0)
link : inAuto inAuto@Tramo2(0,0)
link : inCamino inCamino@Tramo2(1,0)
link : inHayLugar inHayLugar@Tramo2(0,4)
link : inHayLugar inHayLugar@Tramo2(1,4)
out : outQuiereCruzar outAuto outCamino outPuedeEntrar outEntraAuto
link : outPuedeEntrar@tramo2(0,0) outPuedeEntrar
link : outEntraAuto@tramo2(0,0) outEntraAuto
link : outQuiereCruzar@tramo2(0,4) outQuiereCruzar
link : outAuto@tramo2(0,4) outAuto
link : outCamino@tramo2(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,2) }
zone : bacheAuto-rule { (0,3) }
zone : bacheCamino-rule { (1,3) }
zone : DespuesBacheAuto-rule { (0,4) }
zone : DespuesBacheCamino-rule { (1,4) }
portInTransition : inPuedeEntrar@tramo2(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo2(0,0) IngresaAutoTramo2
portInTransition : inCamino@tramo2(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo2(0,4) SaleACruceAuto
```

```
portInTransition : inHayLugar@tramo2(1,4) SaleACruceCamino

[IngresaAutoTramo2]
% Ingresa un auto al tramo (posiblemente desde un cruce)
rule : {portvalue(ThisPort) + send(outEntraAuto, 1)} 10 { portvalue(ThisPort) !=0 and (0,0) = 0 }
rule : 0 10  { t }

[BacheAuto-rule]
% Sale hacia adelante
rule : 0 50 { not isundefined((0,1)) and  (0,0) != 0 and (0,1) = 0 }
% Viene de atras
rule : {(0,-1)} 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) !=0 }
rule : {(0,0)} 10  { t }

[BacheCamino-rule]
% Sale hacia adelante
rule : 0 50 { not isundefined((0,1)) and  (0,0) != 0 and (0,1) = 0 }
% Viene de atras
rule : {(0,-1)} 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) != 0 }
rule : {(0,0)} 10  { t }

[DespuesBacheAuto-rule]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) !=0 and (0,1) = 0 }
% Viene de atras
rule : {(0,-1)} 50 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) !=0 }
% Sale hacia cruce
rule : {(0,0) + send(outQuiereCruzar, 1)} 10 { not isundefined((0,-1)) and isundefined((0,1)) and  (0,0) !=0}
rule : {(0,0)} 10  { t }

[DespuesBacheCamino-rule]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) != 0 and (0,1) = 0 }
% Viene de atras
rule : {(0,-1)} 50 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) != 0 }
rule : {(0,0)} 10  { t }


%*********************************************************************************
%*******************************  TRAMO 3 ****************************************
%*********************************************************************************
[tramo3]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo3(0,-1)  tramo3(0,0)  tramo3(0,1)
neighbors : tramo3(1,-1)  tramo3(1,0)  tramo3(1,1)
initialvalue : 0
initialrow : 0     0 0 0 0 0
initialrow : 1     0 0 0 0 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
link : inPuedeEntrar inPuedeEntrar@Tramo3(0,0)
```

```
link : inAuto inAuto@Tramo3(0,0)
link : inCamino inCamino@Tramo3(1,0)
link : inHayLugar inHayLugar@Tramo3(0,4)
link : inHayLugar inHayLugar@Tramo3(1,4)
out  : outQuiereCruzar outAuto outCamino outPuedeEntrar outEntraAuto
link : outPuedeEntrar@tramo3(0,0) outPuedeEntrar
link : outEntraAuto@tramo3(0,0) outEntraAuto
link : outQuiereCruzar@tramo3(0,4) outQuiereCruzar
link : outAuto@tramo3(0,4) outAuto
link : outCamino@tramo3(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inPuedeEntrar@tramo3(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo3(0,0) IngresaAutoTramo3
portInTransition : inCamino@tramo3(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo3(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo3(1,4) SaleACruceCamino

[IngresaAutoTramo3]
% Ingresa un auto al tramo (posiblemente desde un cruce)
rule : {portvalue(ThisPort) + send(outEntraAuto, 1)} 10 { portvalue(ThisPort) !=0 and (0,0) = 0 }
rule : 0 10  { t }



%*****************************************************************************************
%*******************************  TRAMO 4 ********************************************
%*****************************************************************************************
[tramo4]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo4(0,-1)  tramo4(0,0)  tramo4(0,1)
neighbors : tramo4(1,-1)  tramo4(1,0)  tramo4(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
link : inPuedeEntrar inPuedeEntrar@Tramo4(0,0)
link : inAuto inAuto@Tramo4(0,0)
link : inCamino inCamino@Tramo4(1,0)
link : inHayLugar inHayLugar@Tramo4(0,4)
link : inHayLugar inHayLugar@Tramo4(1,4)
out  : outQuiereCruzar outAuto outCamino outPuedeEntrar
link : outPuedeEntrar@tramo4(0,0) outPuedeEntrar
link : outQuiereCruzar@tramo4(0,4) outQuiereCruzar
link : outAuto@tramo4(0,4) outAuto
link : outCamino@tramo4(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inPuedeEntrar@tramo4(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo4(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo4(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo4(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo4(1,4) SaleACruceCamino
```

```
%*****************************************************************************
%******************************  TRAMO 5 *************************************
%*****************************************************************************
[tramo5]
type : cell
Width : 5
Height : 2
delay : transport
defaultDelayTime  : 10
border : nowrapped
neighbors : tramo5(0,-1)  tramo5(0,0)  tramo5(0,1)
neighbors : tramo5(1,-1)  tramo5(1,0)  tramo5(1,1)
initialvalue : 0
localtransition : tramo-rule
in : inAuto inCamino inHayLugar inPuedeEntrar
link : inPuedeEntrar inPuedeEntrar@Tramo5(0,0)
link : inAuto inAuto@Tramo5(0,0)
link : inCamino inCamino@Tramo5(1,0)
link : inHayLugar inHayLugar@Tramo5(0,4)
link : inHayLugar inHayLugar@Tramo5(1,4)
out  : outQuiereCruzar outAuto outCamino outPuedeEntrar
link : outPuedeEntrar@tramo5(0,0) outPuedeEntrar
link : outQuiereCruzar@tramo5(0,4) outQuiereCruzar
link : outAuto@tramo5(0,4) outAuto
link : outCamino@tramo5(1,4) outCamino
zone : CaminoTramo-rule { (1,0)..(1,4) }
portInTransition : inPuedeEntrar@tramo5(0,0) PuedeEntrarTramo
portInTransition : inAuto@tramo5(0,0) IngresaAutoTramo
portInTransition : inCamino@tramo5(1,0) IngresaCaminoTramo
portInTransition : inHayLugar@tramo5(0,4) SaleACruceAuto
portInTransition : inHayLugar@tramo5(1,4) SaleACruceCamino

%*****************************************************************************
%****************************** REGLAS COMUNES A LOS TRAMOS
************************
%*****************************************************************************
[PuedeEntrarTramo]
% Le indica al cruce si hay lugar para que ingrese el auto
rule : { (0,0) + send(outPuedeEntrar,0) } 10 { (0,0) = 0 }
rule : { (0,0) + send(outPuedeEntrar,1) } 10 { (0,0) != 0 }

[IngresaAutoTramo]
% Ingresa un auto al tramo (posiblemente desde un cruce)
rule : {portvalue(ThisPort)} 10 { portvalue(ThisPort) !=0 and (0,0) = 0 }
rule : 0 10  { t }

[IngresaCaminoTramo]
rule : {portvalue(ThisPort)} 10 {portvalue(ThisPort) != 0 and (0,0) = 0}
rule : 0 10  { t }

[tramo-rule]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) !=0 and (0,1) = 0 }
% Viene de atras
rule : {(0,-1)} 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) !=0 }
```

```
% Sale hacia cruce
rule : {(0,0) + send(outQuiereCruzar, 1)} 10 { not isundefined((0,-1)) and isundefined((0,1)) and  (0,0) != 0}
rule : {(0,0)} 10  { t }

[Caminotramo-rule]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) != 0 and (0,1) = 0 }
% Viene de atras
rule : {(0,-1)} 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) != 0 }
rule : {(0,0)} 10  { t }

[SaleACruceAuto]
% Sale hacia adelante (para la celda de salida)
rule : { 0 + send(outAuto,(0,0) ) } 10 { (0,0) != 0  and portvalue(ThisPort) = 0 }
rule : {(0,0)} 10 { (0,0) != 0  and portvalue(ThisPort) != 0 }
rule : {(0,0)} 10  { t }

[SaleACruceCamino]
% Sale hacia adelante (para la celda de salida)
rule : { 0 + send(outCamino,(0,0)) } 10 { (0,0) != 0 and portvalue(ThisPort) = 0 }
rule : {(0,0)} 10  { t }

%*****************************************************************************
%****************************** CRUCE 1 **************************************
%*****************************************************************************
[cruce1]
type : cell
Width : 3
Height : 5
delay : transport
defaultDelayTime  : 50
border : wrapped
neighbors : cruce1(-4,-1) cruce1(-4,0) cruce1(-4,1)
neighbors : cruce1(-3,-1) cruce1(-3,0) cruce1(-3,1)
neighbors : cruce1(-2,-1) cruce1(-2,0) cruce1(-2,1)
neighbors : cruce1(-1,-1) cruce1(-1,0) cruce1(-1,1)
neighbors : cruce1(0,-1) cruce1(0,0) cruce1(0,1)
neighbors : cruce1(1,-1) cruce1(1,0) cruce1(1,1)
neighbors : cruce1(2,-1) cruce1(2,0) cruce1(2,1)
neighbors : cruce1(3,-1) cruce1(3,0) cruce1(3,1)
neighbors : cruce1(4,-1) cruce1(4,0) cruce1(4,1)
initialvalue : 0
localtransition : AutoCruce1-entrada
in : inAuto inCamino inHayAuto inHayLugar2 inHayLugar3 inPeso inNuevoCamino
in : inNuevoCaminoAuto inPesoCamino inPesoRuteado
link : inAuto inAuto@Cruce1(0,0)
link : inCamino inCamino@Cruce1(1,0)
link : inAuto inAuto@Cruce1(4,0)
link : inNuevoCamino inNuevoCamino@Cruce1(0,0)
link : inNuevoCamino inNuevoCamino@Cruce1(1,0)
link : inNuevoCamino inNuevoCamino@Cruce1(4,0)
link : inHayAuto inPregunta@Cruce1(0,0)
link : inPeso inPeso@Cruce1(0,0)
link : inPesoCamino inPeso@Cruce1(1,0)
link : inPesoRuteado inPeso@Cruce1(4,0)
link : inHayLugar2 inHayLugar@Cruce1(0,1)
```

```
link : inHayLugar2 inHayLugar@Cruce1(1,1)
link : inHayLugar2 inHayLugar@Cruce1(4,1)
link : inHayLugar3 inHayLugar@Cruce1(0,2)
link : inHayLugar3 inHayLugar@Cruce1(1,2)
link : inHayLugar3 inHayLugar@Cruce1(4,2)
out : outHayLugarParaEntrar outPuedeSalir2 outAuto2 outCamino2 outPuedeSalir3 outAuto3 outCamino3
out : outSolicitaPeso2 outSolicitaPeso3 outSolicitaCamino
link : haylugar@cruce1(0,0) outHayLugarParaEntrar
link : outSolicitaPeso2@cruce1(1,0) outSolicitaPeso2
link : outSolicitaPeso3@cruce1(1,0) outSolicitaPeso3
link : outSolicitaCamino@cruce1(1,0) outSolicitaCamino
link : PuedeSalir@cruce1(0,1) outPuedeSalir2
link : auto@cruce1(0,1) outAuto2
link : camino@cruce1(1,1) outCamino2
link : PuedeSalir@cruce1(0,2) outPuedeSalir3
link : auto@cruce1(0,2) outAuto3
link : camino@cruce1(1,2) outCamino3
initialrowvalue : 2     111
initialrowvalue : 3     023
zone : AutoCruce1-entrada { (0,0) }
zone : AutoCruce1-salida { (0,1)..(0,2) }
zone : CaminoCruce1-entrada { (1,0) }
zone : CaminoCruce1-salida { (1,1)..(1,2) }
zone : idcruce-rule { (2,0)..(2,2) }
zone : idtramo-rule { (3,0)..(3,2) }
zone : RuteadoCruce1-entrada { (4,0) }
zone : RuteadoCruce1-salida { (4,1)..(4,2) }
portInTransition : inAuto@cruce1(0,0) IngresaAutoCruce1
portInTransition : inCamino@cruce1(1,0) IngresaCaminoCruce1
portInTransition : inAuto@cruce1(4,0) IngresaRuteadoCruce1
portInTransition : inPregunta@cruce1(0,0) QuiereCruzar
portInTransition : inHayLugar@cruce1(0,1) PuedeSalirAutoCruce1
portInTransition : inHayLugar@cruce1(0,2) PuedeSalirAutoCruce1
portInTransition : inHayLugar@cruce1(1,1) PuedeSalirCaminoCruce1
portInTransition : inHayLugar@cruce1(1,2) PuedeSalirCaminoCruce1
portInTransition : inHayLugar@cruce1(4,1) PuedeSalirRuteadoCruce1
portInTransition : inHayLugar@cruce1(4,2) PuedeSalirRuteadoCruce1
portInTransition : inPeso@cruce1(0,0) IngresaPesoAuto
portInTransition : inPeso@cruce1(1,0) IngresaPesoCamino
portInTransition : inPeso@cruce1(4,0) IngresaPesoRuteado
portInTransition : inNuevoCamino@cruce1(0,0) IngresaNuevoCaminoAuto
portInTransition : inNuevoCamino@cruce1(1,0) IngresaNuevoCamino
portInTransition : inNuevoCamino@cruce1(4,0) IngresaNuevoCaminoRuteado

[AutoCruce1-entrada]
% Elimina el auto de la simulación si ha llegado a destino
rule : 0 10 { (0,0)!=0 and #Macro(Es_Destino) }
% Ingresa a la celda un auto que ya estaba dentro del cruce
% (sólo si ya se encuentra ruteado)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) !=0 and (4,-1) = 1 }
% Avanza a la prox celda (sólo si está ruteado)
rule : 0 10 { (0,0) != 0 and (0,1) = 0 and (4,0) = 1 }
rule : {(0,0)} 10  { t }

[CaminoCruce1-entrada]
% Elimina el camino de la simulación si ha llegado a destino
```

```
rule : 0 10 { (0,0) != 0 and #Macro(Es_Destino) }
% Ingresa a la celda un camino que ya estaba dentro del cruce
% (sólo si ya se encuentra ruteado)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) > 0 and (3,-1) = 1}
% Avanza a la prox celda (sólo si está ruteado)
rule : 0 10 { (0,0) != 0 and (0,1) = 0 and (3,0) = 1 }
rule : {(0,0)} 10  { t }

[RuteadoCruce1-entrada]
% Elimina el auto de la simulación si ha llegado a destino (la segunda condición indica si ha llegado a
destino)
rule : 0 10 { (-4,0) != 0 and (-3,0) = -1 }
% Ingresa a la celda un auto que ya estaba dentro del cruce
% (sólo si ya se encuentra ruteado) *********
rule : {(0,-1)} 10 { (-4,0) = 0 and (-4,-1) !=0 and (0,-1) = 1 }
% Avanza a la prox celda
rule : 0 10 { (-4,0) != 0 and (-4,1) = 0 and (0,0) = 1}
rule : {(0,0)} 10  { t }


[AutoCruce1-salida]
% Avanza el auto si no debe salir por el tramo (de acuerdo al camino)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) != 0 and (4,-1) = 1 and not #Macro(SalirAuto) }
% Le indica al tramo que hay un auto que puede entrar al mismo (si corresponde al camino)
rule : { 0 + send(PuedeSalir,1)} 10 { (0,0) = 0 and (0,-1) != 0 and (4,-1) = 1 and #Macro(SalirAuto) }
% Avance de un auto dentro del cruce
rule : 0 10 { (0,0) != 0 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[CaminoCruce1-salida]
% Avanza el camino si no debe salir por el tramo (de acuerdo al camino)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) != 0 and (3,-1) = 1 and not #Macro(SalirCamino) }
% Avance de un camino dentro del cruce
rule : 0 10 { (0,0)!=0 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[RuteadoCruce1-salida]
% Avanza el auto si no debe salir por el tramo (de acuerdo al camino)
rule : {(0,-1)} 10 { (-4,0) = 0 and (-4,-1) != 0 and (0,-1) = 1 and not (Trunc((-3,-1)) = trunc((-1,0)) ) }
% Le indica al tramo que hay un auto que puede entrar al mismo (si corresponde al camino)
rule : 0 10 { (-4,0) = 0 and (-4,-1) != 0 and (0,-1) = 1 and (Trunc((3,-1)) = trunc((1,0)) ) }
% Avance de un auto dentro del cruce
rule : 0 10 { (-4,0) != 0 and (-4,1) = 0 }
rule : {(0,0)} 10  { t }

[IngresaAutoCruce1]
% Nota: no puedo decidir acá si llegó o no a destino ya que no puedo leer
% el port que contiene el camino
rule : {portvalue(ThisPort)} 0 { (0,0) = 0 and (0,-1) = 0 and portvalue(ThisPort) !=0 }
rule : {(0,0)} 10  { t }

[IngresaCaminoCruce1]
rule : {portvalue(thisPort) + send(outSolicitaPeso2,1)} 0 { (0,0) = 0 and (0,-1) = 0 and portvalue(ThisPort) !=
0  and #Macro(ProximoTramo) = 2}
rule : {portvalue(thisPort) + send(outSolicitaPeso3,1)} 0 { (0,0) = 0 and (0,-1) = 0 and portvalue(ThisPort) !=
0  and (#Macro(ProximoTramo) = 3)}
```

```
rule : {(0,0)} 10 { t }

[IngresaRuteadoCruce1]
rule : 0 0 { (-4,0) = 0 and (-4,-1) = 0 and portvalue(ThisPort) !=0 }
rule : {(0,0)} 10  { t }


[IngresaPesoAuto]
rule : {(0,0)} 10 { t }

[IngresaPesoCamino]
%Detecta congestion en el tramo
rule : { (0,0) + send(outSolicitaCamino, #Macro(ValorAOD)) } 10 {portvalue(ThisPort) >= 2}
rule :  {(0,0)} 10 { t }

[IngresaPesoRuteado]
%Detecta congestion en el tramo
rule : 1 10 {portvalue(ThisPort) < 2}
rule : {(0,0)} 10  { t }

[IngresaNuevoCamino]
rule : {portvalue(thisPort)} 10 {portvalue(ThisPort) !=0}
rule : {(0,0)} 10 { t }

[IngresaNuevoCaminoAuto]
rule : {(0,0)} 10 { t }

[IngresaNuevoCaminoRuteado]
rule : 1 10 {portvalue(ThisPort) !=0}
rule : {(0,0)} 10 { t }


[PuedeSalirAutoCruce1]
% Llega auto que abandonara el cruce
rule : {0 + send(auto,(0,-1))} 10 { (0,0) = 0 and (0,-1) != 0 and (4,-1) = 1 and portvalue(ThisPort) = 0 and
#Macro(SalirAuto) }
% Llega auto que permanecera dentro del cruce
rule : {(0,-1) + send(auto, 0)} 10 { (0,0) = 0 and (0,-1) != 0 and (4,-1) = 1 and ( portvalue(ThisPort) = 1 or (
portvalue(ThisPort) = 0 and not #Macro(SalirAuto) )) }
rule : {(0,0)} 10  { t }

[PuedeSalirCaminoCruce1]
% Llega auto que abandonara el cruce (no es el último tramo)
% Nota: se reemplazó la condición #Macro(NuevoCamino) != 0 por #Macro(NuevoCamino) >= 0.0001 por
problemas de redondeo.
% Esta condición funciona siempre y cuando la identificación de los tramos tenga hasta 4 dígitos.
rule : {0 + send(camino,#Macro(NuevoCamino))} 10 { (0,0) = 0 and (0,-1) != 0 and (3,-1) = 1 and
portvalue(ThisPort) = 0 and #Macro(SalirCamino) and #Macro(NuevoCamino) >= 0.0001}
% Llega auto que abandonara el cruce (es el último tramo)
rule : {0 + send(camino,-1)} 10 { (0,0) = 0 and (0,-1) != 0 and (3,-1) = 1 and portvalue(ThisPort) = 0 and
#Macro(SalirCamino) and #Macro(NuevoCamino) < 0.0001}
% Llega camino que permanecera dentro del cruce
rule : {(0,-1) + send(camino, 0)} 10 { (0,0) = 0 and (0,-1) != 0 and (3,-1) = 1 and ( portvalue(ThisPort) = 1 or
( portvalue(ThisPort) = 0 and not #Macro(SalirCamino) )) }
rule : {(0,0)} 10  { t }
```

```
[PuedeSalirRuteadoCruce1]
% Llega auto que abandonara el cruce
rule : 0 10 { (-4,0) = 0 and (-4,-1) != 0 and (0,-1) = 1 and portvalue(ThisPort) = 0 and (Trunc((-3,-1)) =
trunc((-1,0)) ) }
% Llega auto que permanecera dentro del cruce
rule : {(0,-1)} 10 { (-4,0) = 0 and (-4,-1) != 0 and (0,-1) = 1 and ( portvalue(ThisPort) = 1 or (
portvalue(ThisPort) = 0 and not  (Trunc((-3,-1)) = trunc((-1,0)) ) )) }
rule : {(0,0)} 10  { t }


%*****************************************************************************************
%******************************** CRUCE 2 ********************************************
%*****************************************************************************************

[cruce2]
type : cell
Width : 2
Height : 4
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : cruce2(0,-1) cruce2(0,0) cruce2(0,1)
neighbors : cruce2(1,-1) cruce2(1,0) cruce2(1,1)
neighbors : cruce2(2,-1) cruce2(2,0) cruce2(2,1)
neighbors : cruce2(3,-1) cruce2(3,0) cruce2(3,1)
initialvalue : 0
localtransition : autoCruce-entrada
in : inAuto inCamino inHayAuto inHayLugar
link : inAuto inAuto@Cruce2(0,0)
link : inCamino inCamino@Cruce2(1,0)
link : inHayAuto inPregunta@Cruce2(0,0)
%link : inHayAuto inPregunta@Cruce2(1,0)
link : inHayLugar inHayLugar@Cruce2(0,1)
link : inHayLugar inHayLugar@Cruce2(1,1)
out : outHayLugarParaEntrar outPuedeSalir5 outAuto outCamino
link : haylugar@cruce2(0,0) outHayLugarParaEntrar
link : haylugar@cruce2(1,0) outHayLugarParaEntrar
link : PuedeSalir@cruce2(0,1) outPuedeSalir5
link : auto@cruce2(0,1) outAuto
link : camino@cruce2(1,1) outCamino
initialrowvalue : 2     22
initialrowvalue : 3     05
zone : AutoCruce-salida { (0,1) }
zone : CaminoCruce-entrada { (1,0) }
zone : CaminoCruce-salida { (1,1) }
zone : idcruce-rule { (2,0)..(2,1) }
zone : idtramo-rule { (3,0)..(3,1) }
portInTransition : inAuto@cruce2(0,0) IngresaAutoCruce
portInTransition : inCamino@cruce2(1,0) IngresaCaminoCruce
portInTransition : inPregunta@cruce2(0,0) QuiereCruzar
portInTransition : inHayLugar@cruce2(0,1) PuedeSalirAutoCruce
portInTransition : inHayLugar@cruce2(1,1) PuedeSalirCaminoCruce

%*****************************************************************************************
%******************************** CRUCE 3 ********************************************
%*****************************************************************************************
```

```
[cruce3]
type : cell
Width : 2
Height : 4
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : cruce3(0,-1) cruce3(0,0) cruce3(0,1)
neighbors : cruce3(1,-1) cruce3(1,0) cruce3(1,1)
neighbors : cruce3(2,-1) cruce3(2,0) cruce3(2,1)
neighbors : cruce3(3,-1) cruce3(3,0) cruce3(3,1)
initialvalue : 0
localtransition : autoCruce-entrada
in : inAuto inCamino inHayAuto inHayLugar
link : inAuto inAuto@Cruce3(0,0)
link : inCamino inCamino@Cruce3(1,0)
link : inHayAuto inPregunta@Cruce3(0,0)
%link : inHayAuto inPregunta@Cruce3(1,0)
link : inHayLugar inHayLugar@Cruce3(0,1)
link : inHayLugar inHayLugar@Cruce3(1,1)
out : outHayLugarParaEntrar outPuedeSalir4 outAuto outCamino
link : haylugar@cruce3(0,0) outHayLugarParaEntrar
link : haylugar@cruce3(1,0) outHayLugarParaEntrar
link : PuedeSalir@cruce3(0,1) outPuedeSalir4
link : auto@cruce3(0,1) outAuto
link : camino@cruce3(1,1) outCamino
initialrowvalue : 2     33
initialrowvalue : 3     04
zone : AutoCruce-salida { (0,1) }
zone : CaminoCruce-entrada { (1,0) }
zone : CaminoCruce-salida { (1,1) }
zone : idcruce-rule { (2,0)..(2,1) }
zone : idtramo-rule { (3,0)..(3,1) }
portInTransition : inAuto@cruce3(0,0) IngresaAutoCruce
portInTransition : inCamino@cruce3(1,0) IngresaCaminoCruce
portInTransition : inPregunta@cruce3(0,0) QuiereCruzar
portInTransition : inHayLugar@cruce3(0,1) PuedeSalirAutoCruce
portInTransition : inHayLugar@cruce3(1,1) PuedeSalirCaminoCruce


%*****************************************************************************
%******************************** CRUCE 4 ************************************
%*****************************************************************************
[cruce4]
type : cell
Width : 2
Height : 4
delay : transport
defaultDelayTime  : 10
border : wrapped
neighbors : cruce4(0,-1) cruce4(0,0)  cruce4(0,1)
neighbors : cruce4(1,-1) cruce4(1,0)  cruce4(1,1)
neighbors : cruce4(2,-1) cruce4(2,0)  cruce4(2,1)
neighbors : cruce4(3,-1) cruce4(3,0)  cruce4(3,1)
initialvalue : 0
```

```
localtransition : AutoCruce-entrada
in : inAuto4 inAuto5 inCamino4 inCamino5 inHayAuto4 inHayAuto5
link : inAuto4 inAuto@Cruce4(0,0)
link : inCamino4 inCamino@Cruce4(1,0)
link : inAuto5 inAuto@Cruce4(0,1)
link : inCamino5 inCamino@Cruce4(1,1)
link : inHayAuto4 inPregunta@Cruce4(0,0)
link : inHayAuto5 inPregunta@Cruce4(0,1)
out : outHayLugarParaEntrar4 outHayLugarParaEntrar5
link : haylugar@cruce4(0,0) outHayLugarParaEntrar4
link : haylugar@cruce4(0,1) outHayLugarParaEntrar5
initialrowvalue : 2     44
initialrowvalue : 3     00
zone : CaminoCruce-entrada { (1,0)..(1,1) }
zone : idcruce-rule { (2,0)..(2,1) }
zone : idtramo-rule { (3,0)..(3,1) }
portInTransition : inAuto@cruce4(0,0) IngresaAutoCruce
portInTransition : inCamino@cruce4(1,0) IngresaCaminoCruce
portInTransition : inAuto@cruce4(0,1) IngresaAutoCruce
portInTransition : inCamino@cruce4(1,1) IngresaCaminoCruce
portInTransition : inPregunta@cruce4(0,0) QuiereCruzar
portInTransition : inPregunta@cruce4(0,1) QuiereCruzar



%*****************************************************************************************
%******************************** REGLAS COMUNES A LOS CRUCES
************************
%*****************************************************************************************
[QuiereCruzar]
rule : { (0,0) + send(haylugar,0) } 10 { (0,0) = 0 and (0,-1) = 0 and portvalue(ThisPort) = 1 }
rule : { (0,0) + send(HayLugar,1) } 10  { ((0,0) !=0 or ((0,0) = 0 and (0,-1) !=0)) and portvalue(ThisPort) = 1
}
rule : {(0,0)} 10  { t }

[AutoCruce-entrada]
% Elimina el auto de la simulación si ha llegado a destino
rule : 0 10 { (0,0)!=0 and #Macro(Es_Destino) }
% Ingresa a la celda un auto que ya estaba dentro del cruce
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) !=0 and (1,-1) != -1 }
% Avanza a la prox celda
rule : 0 10 { (0,0) != 0 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[CaminoCruce-entrada]
% Elimina el camino de la simulación si ha llegado a destino
rule : 0 10 { (0,0) != 0 and (0,0) = -1 }
% Ingresa a la celda un camino que ya estaba dentro del cruce
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1) > 0 }
% Avanza a la prox celda
rule : 0 10 { (0,0) != 0 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[AutoCruce-salida]
% Le indica al tramo que hay un auto que puede entrar al mismo (si corresponde al camino)
rule : { 0 + send(PuedeSalir,1)} 10 { (0,0) = 0 and (0,-1)!=0 and #Macro(SalirAuto) }
```

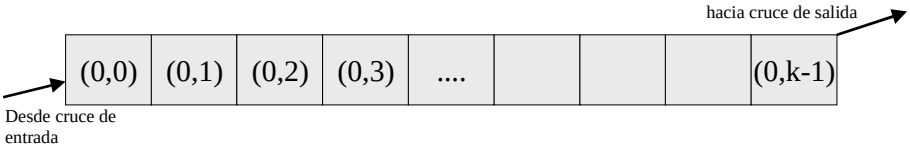
```
% Avanza el auto si no debe salir por el tramo (de acuerdo al camino)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1)!=0 and not #Macro(SalirAuto) }
% Avance de un auto dentro del cruce
rule : 0 10 { (0,0)!=0 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[CaminoCruce-salida]
% Avanza el camino si no debe salir por el tramo (de acuerdo al camino)
rule : {(0,-1)} 10 { (0,0) = 0 and (0,-1)!=0 and not #Macro(SalirCamino) }
% Avance de un camino dentro del cruce
rule : 0 10 { (0,0)!=0 and (0,1) = 0 }
rule : {(0,0)} 10  { t }

[IngresaAutoCruce]
rule : {portvalue(ThisPort)} 10 { (0,0) = 0 and portvalue(ThisPort) != 0 }
rule : {(0,0)} 10 { t }

[IngresaCaminoCruce]
rule : {portvalue(thisPort) } 10 { (0,0) = 0 and portvalue(ThisPort) != 0}
rule : {(0,0)} 10 { t }

[PuedeSalirAutoCruce]
% Llega auto que abandonara el cruce
rule : {0 + send(auto, (0,-1)) } 10 { (0,0) = 0 and (0,-1)!=0 and portvalue(ThisPort) = 0 and
#Macro(SalirAuto) }
% Llega auto que permanecera dentro del cruce
rule : {0 + send(auto, 0)} 10 { (0,0) = 0 and  (0,-1)!=0 and ( portvalue(ThisPort) = 1 or ( portvalue(ThisPort)
= 0 and not #Macro(SalirAuto) )) }
rule : {(0,0)} 10  { t }

[PuedeSalirCaminoCruce]
% Llega auto que abandonara el cruce (no es el último tramo)
% Nota: se reemplazó la condición #Macro(NuevoCamino) != 0 por #Macro(NuevoCamino) >= 0.0001 por
problemas de redondeo.
% Esta condición funciona siempre y cuando la identificación de los tramos tenga hasta 4 dígitos.
rule : {0 + send(camino,#Macro(NuevoCamino))} 10 { (0,0) = 0 and (0,-1) != 0 and portvalue(ThisPort) = 0
and #Macro(SalirCamino) and #Macro(NuevoCamino) >= 0.0001}
% Llega auto que abandonara el cruce (es el último tramo)
rule : {0 + send(camino,-1)} 10 { (0,0) = 0 and (0,-1) != 0 and portvalue(ThisPort) = 0 and
#Macro(SalirCamino) and #Macro(NuevoCamino) < 0.0001}
% Llega camino que permanecera dentro del cruce
rule : {(0,-1) + send(camino, 0)} 10 { (0,0) = 0 and (0,-1) != 0 and ( portvalue(ThisPort) = 1 or (
portvalue(ThisPort) = 0 and not #Macro(SalirCamino) )) }
rule : {(0,0)} 10  { t }

[idcruce-rule]
rule : {(0,0)} 10  { t }

[idtramo-rule]
rule : {(0,0)} 10  { t }
```

1.3.2. Archivo : rd.inc

```
#BeginMacro(SalirAuto)
 (Trunc((1,-1)) = trunc((3,0)) )
#EndMacro

#BeginMacro(SalirCamino)
 (Trunc((0,-1)) = trunc((2,0)) )
#EndMacro

#BeginMacro(Es_destino)
 ( (1,0) = -1 )
#EndMacro

#BeginMacro(NuevoCamino)
 ( Fractional((0,-1)) * 10 )
#EndMacro

#BeginMacro(NuevoCamino1)
 ( Fractional((0,0)) * 10 )
#EndMacro

#BeginMacro(ProximoTramo)
 Trunc(portvalue(thisPort))
#EndMacro

#BeginMacro(ValorAOD)
 ( Trunc((0,0)) + ( Trunc((1,0)) * 0.01) + ( Trunc((-1,0)) * 0.0001) )
#EndMacro
```

Apéndice C: Problemas encontrados en N-CD++

1) Los estados de las celdas sólo soportan números reales. Los modelos aquí presentados requieren una representación más compleja.
   En la especificación de los modelos se definió el estado de una celda como de tipo 'Record'. Para poder representar esto en la herramienta, un modelo unidimensional con un estado de tipo 'Record' se transformó en un modelo bidimensional donde cada componente del tipo 'Record' se almacenó en una fila del modelo celular.
   Para ilustrar esto mostramos el siguiente ejemplo:
   Se tiene un tramo de un carril (por lo tanto, se define un modelo de una dimensión) compuesto por 5 celdas (k=5), cuyo estado es s = (*destino*, *camino*).



   En N-CD++ se implementó de la siguiente forma

```
Width : 5
Height : 2
Neighbors : tramo1(0,-1)  tramo1(0,0)  tramo1(0,1)
Neighbors : tramo1(1,-1)  tramo1(1,0)  tramo1(1,1)
```

   En lo que respecta a la herramienta, las celdas de una misma columna son celdas independientes, mientras que para nosotros es una sola celda.

   Como consecuencia de esta modificación, fue necesario redefinir:
   - El vecindario de las celdas: Se incluyen como vecinos todos los niveles que representan los distintos componentes del registro.
   - Las reglas del modelo: Esto requirió redefinir las reglas de tal forma que, los movimientos hechos por el vehículo estén sincronizados en todos los niveles (por ejemplo, el destino y el camino del auto deben actuar en forma perfectamente sincronizada).
     Ejemplo

Reglas en el modelo de especificación		
Nuevo estado	**Estado del vecindario**	**Nombre de la regla**
Destino = Destino(0, -1) *Camino* = Camino(0,-1)	Destino(0,-1) != 0 And Destino(0,0) = 0	*Llega_Desde_Atrás*
Destino = 0 *Camino* = 0	Destino(0,0) != 0 and Destino(0,1) = 0	*Sale_Hacia_Adelante*
(0,0)	t /*en otro caso conserva estado*/	*Default*

Definición en N-CD++

```
/* Reglas correspondientes al nivel que representa al destino */
[tramo-rule]
```

```
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) !=0 and (0,1) = 0 }
% Llega desde atras
rule : {(0,-1)} 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) !=0 }
% Default
rule : {(0,0)} 10  { t }

[Caminotramo-rule]
% Sale hacia adelante
rule : 0 10 { not isundefined((0,1)) and  (0,0) != 0 and (0,1) = 0 }
% Llega desde atras
rule : {(0,-1)} 10 { not isUndefined((0,-1)) and (0,0) = 0  and (0,-1) != 0 }
rule : {(0,0)} 10  { t }
```

Como puede verse en este ejemplo, las reglas debieron ser separadas en dos secciones independientes, una para cada nivel, teniendo que prestar especial atención a la sincronización de las mismas. Esto a medida que se quieren construir ejemplos más complejos constituyen una fuente de error considerable en la construcción de los modelos.

- La interconexión de los ports: Para que todos los niveles actúen en forma sincronizada los ports deben estar conectados a cada uno de ellos.

```
/* El port de entrada al tramo que indica si hay lugar en el cruce para que el auto pueda salir hacia él, debe estar conectado tanto al nivel que contiene al destino como al camino, de manera que se envíen ambos datos. */
in : inHayLugar
link : inHayLugar inHayLugar@Tramo1(0,4)
link : inHayLugar inHayLugar@Tramo1(1,4)
```

2) La herramienta posee un manejo distinto para la lectura de los ports.
En la definición del formalismo, al enviar un modelo información a través de un port de salida, todos los demás ports que están conectados a él, tienen la información disponible en cualquier instante. De esta forma, cualquier regla puede leer el valor del port.
En la herramienta, los ports de entrada sólo pueden ser leídos en secciones especiales denominadas 'portintransition', las cuales poseen reglas que se ejecutarán cuando le llegue un valor al mismo. Esto implica que sólo serán ejecutadas cuando se produzca el evento de entrada (no en cualquier momento).
Por ello, todas las reglas del modelo que leían ports de entrada debieron ser modificadas. En lugar de leer el port, ahora le 'piden' al modelo al cuál están conectadas (a través de un port de salida) que le envíe el valor del mismo. Esto produce que, al recibir la respuesta se ejecute la sección portintransition asociada al port que contiene la regla original del modelo.
Este cambio de concepto requirió:
- La definición de nuevos ports: Los que 'piden' un valor (port de salida) y los que reciben el pedido (port de entrada) y que provocan el envío.
- La redefinición de las reglas.
- En algunos casos, el cambio de especificación, ya que la celda debe quedar esperando la ocurrencia de un evento (éste fue el caso de la implementación del ruteo dinámico al evaluar la existencia de congestión y el pedido de un camino alternativo)

3) Es muy trabajosa y propensa a errores, la definición de modelos complejos. Prueba de esto, son los ejemplos de ruteo presentados, cuya funcionalidad es sencilla, pero su implementación es extensa y engorrosa.
   Este problema puede superarse con la implementación del lenguaje de implementación que automatice el mapeo con el formalismo Cell-Devs.

4) Se han presentado algunos problemas con la herramienta, los cuales están siendo analizados:
   4.1) Existen problemas relacionados con la demora con la cual se inyectan autos en la simulación. Si los vehículos ingresan a la simulación separados por intervalos constantes y mayores a las demoras de las celdas, el modelo se comporta según lo esperado, pero en caso contrario su comportamiento es errático. Este problema no permitió que se inyectaran autos al ejemplo del sector de ciudad con distribución aleatoria.
   4.2) Cuando un vehículo quiere salir de un tramo hacia un cruce, y no lo puede hacer por estar la celda del cruce ocupada, debe planificar un evento para reintentar el movimiento. Este comportamiento no se está produciendo, ya que no se encola la planificación del evento descripto. Se está analizando la causa del problema.
   4.3) Se presentan problemas de sincronización ante la ocurrencia de eventos simultáneos, hecho totalmente probable en los modelos presentados en este trabajo, ya que los niveles que representan parte del estado de la celda necesitan actualizarse en forma sincronizada.
   4.4) Se han detectado problemas de redondeo, que en determinadas circunstancias causan decisiones erróneas en el ruteo de los vehículos (actualización incorrectas de los caminos y errores en la reglas de los cruces que determinan si se debe salir o no por un tramo).

Apéndice D: Contenido del CD-Rom

El contenido del CD-Rom será el siguiente

Directorios		**Descripción**
Modelos Devs	Congest *OD*	Contiene el código de los modelos DEVs implementados (en C++): Modelo Congestión y OD, para ser utilizados por la herramienta N-CD++.
Ejemplos	Ciudad Ruteo estático Ruteo dinámico	Contiene el código de los ejemplos implementados en la herramienta N-CD++.
Corridas	Ciudad – Corrida1_HoraNoPico Ciudad – Corrida2_HoraPico Ciudad – Corrida3_Bache Ciudad – Corrida4_Via Ruteo dinámico Ruteo estático	Contiene las corridas realizadas de los ejemplos de la Ciudad, ruteo dinámico y ruteo estático.
Doc		Contiene el informe de la Tesis.

Apéndice E: Funciones utilizadas en la especificación de los modelos descriptos.

- ***Proximo_Tramo(c: Camino): Tramo***
  Esta función recibe como parámetro un camino y devuelve el primer tramo del mismo.

- ***Cola_Camino(c:Camino): Camino***
  Esta función retorna el mismo camino pasado como parámetro, pero sin el primer tramo. Se usa para ir consumiendo el camimo a medida que se transita por los tramos.

- ***Nuevo_Camino (Origen: N, Destino: N, CaminoActual : Camino) : Camino***
  Dado el cruce de partida (origen), el cruce al cual se quiere llegar (Destino) y el camino actual que sigue el auto, devuelve un camino alternativo que conduzca del origen al destino. La elección del camino alternativo puede definirse dentro de la función, dando mayor flexibilidad a este modelo.
  Si se cuenta con una matriz OD del tipo 'Tabla de Caminos' (ver sección IV) en la que registro, una alternativa sería buscar en la matriz el camino cuyo origen y destino sean los pasados como parámetro, con mejor tiempo de viaje y que difiera del parámetro camino actual.

- ***Hay_Congestion (Valor_congestión: N, Identificación del tramo: N)***
  Devuelve un valor booleano que indica si el tramo está muy congestionado. Una posible definición puede ser que un x% del tramo se encuentre ocupado.

- ***Velocidad:*** Se utilizan en el informe dos funciones 'Velocidad' que difieren en los parámetros que reciben

  Velocidad (velocidad_actual: R, tipo_vehiculo: N, tipo_conductor: N,
  distancia_auto_cercano: R, velocidad_auto_cercano: R,
  velocidad_maxima_tramo: R)

  Es una función que calcula la nueva velocidad del auto basándose en la velocidad actual del mismo, al tipo de vehículo, el tipo de conductor, la distancia y velocidad del auto más cercano hacia adelante y la velocidad máxima permitida en el mismo.

  Velocidad (velocidad_actual: R, n: N, k:N, peso:R, tipo_vehiculo: N,
  tipo_conductor: N, factor_aceleracion: N, Velocidad_maxima_tramo:R)

  Es una función que calcula la nueva velocidad del auto basándose en la velocidad actual del mismo, a la cantidad de celdas del tramo (k), el número de carriles del mismo (n), la congestión que presenta (dado por el parámetro peso), el tipo de vehículo, el tipo de conductor, el factor correctivo anteriormente explicado y la velocidad máxima permitida en el tramo.

- ***IF_ELSE***

La función IF_ELSE se define de la siguiente forma:
IF_ELSE(expresión, condición, valor1, valor2)
IF *expresión* = *condición* THEN
  RETURN *valor1*
ELSE
  RETURN *valor2*
END IF

- ***Distancia ({ Destino(0, i), 1≤i≤v })*** donde v es un parámetro de la simulación e indica la visibilidad que tiene el conductor.
  Esta función se utiliza para cuál es el auto más cercano a una celda. 'Distancia' recibe como parámetro una lista con los estados (indica sólo el destino de los autos contenidas en ellas, si el destino es igual a 0 quiere decir que la celda está libre) de todas las celdas de la vecindad que tiene delante, y devuelve el nro de la primer celda ocupada.

- *Ctd_Celdas (t: Tramos): ℕ*
  Esta función establece la cantidad de celdas que contendrá el tramo t; calculando su longitud (a partir de la distancia entre sus extremos), y luego dividiéndola por el tamaño definido para la celda.
  Sea t = (c1, c2, n, a, dir, max) untramo
  Si a = 0 entonces ℓ = Long_Recta(c1,c2) /* Long_Recta se define más adelante */
  Sino ℓ = Perímetro(c1,c2) /* Perímetro se define más adelante */
  k = ⌈ ℓ / long_celda ⌉
  devolver (k)
  Fin Ctd_Celdas.

Esta función es idéntica a la definida en [DW99]

- ***Conversión_Demora(veloc: ℕ): ℕ***
  Es una función que recibe como parámetro un valor natural que representa una velocidad (en km/h) y devuelve el tiempo (en segundos) que se debe demorar el auto en la celda para representar que avanza con esa velocidad.
  Se define como:

$$\text{Conversión\_Demora(veloc)} = \frac{long\_celda(km) * (60)^2}{veloc(km/h)}$$

  Esta función divide la distancia que se debe recorrer (la longitud de una celda) sobre la velocidad utilizada para hacerlo, obteniendo el tiempo necesario o demora. El factor de $(60)^2$ se utiliza para devolver la demora en segundos.

- ***Ports_In_Out((c, maxc): Cruce, T: cjto de Tramos): { (t,i) / t ∈ Tramos, i ∈ ℕ}, { (t,i) / t ∈ Tramos, i ∈ ℕ}***
  Esta función recibe como parámetros un cruce (c, maxc) y el conjunto de tramos T, y devuelve dos conjuntos. El primero contendrá los tramos que llegan al cruce (con dirección de circulación de vehículos hacia c) y la posición (i) en que el primer carril se conecta al mismo. El segundo contendrá los tramos que salen desde cruce (con dirección de circulación opuesta a c) y la posición (i) en que el primer carril se conecta al mismo.

Para establecer la posición en que cada tramo se acopla al cruce se define un ordenamiento según el ángulo de inclinación de los mismos respecto a la recta y = y1 con c = (x1,y1). Así, a medida que los tramos tengan mayor ángulo les corresponderán las posiciones mayores dentro del cruce. Logrando que los tramos cercanos ocupen las celdas adyacentes dentro del cruce.

Por la restricción impuesta en el lenguaje de especificación a lo sumo puede haber dos tramos que se acoplan al mismo cruce con igual inclinación pero tendrán distinta dirección. Para establecer un ordenamiento total de los tramos, en el caso de tener igual ángulo se adopta la convención de primero acoplar el tramo con dirección hacia c.

Para construir los conjuntos que devuelve esta función, primero se obtienen los tramos del conjunto T que tienen como algún extremo a c:

T' = { t / t ∈ T ∧ t = (c1,c2, n, a, dir, max) ∧ (c1 = c ∨ c2 = c) }

Luego se evalúan los ángulos de inclinación de estos tramos y en forma creciente se les asignan las posiciones del cruce por las que se acoplarán. Esta información se guarda en el conjunto V que contiene tuplas de la forma (t,i,α), que representan que el primer carril del tramo t se conecta al cruce (c, maxc) por la celda i y que α es el ángulo de inclinación de t. Este conjunto se describe como:

$$V = \left\{ \begin{array}{l} ((c_1,c_2,n,a,dir,max),i,\alpha) / (c_1,c_2,n,a,dir,max) \in T' \wedge i \text{ se define en (1)} \wedge \\ \qquad \left[(c \neq c_1 \wedge \alpha = Inclinación(c,c_1)) \vee (c \neq c_2 \wedge \alpha = Inclinación(c,c_2)) \right] \end{array} \right\}$$

$$(1)\ i = \begin{cases} \sum_{(c_1',c_2',n',a',dir',max') \in T' \wedge Inclinación(c_1',c_2') < \alpha} n' & si \quad (dir = 1 \Rightarrow c_2 = c) \wedge (dir = 0 \Rightarrow c_1 = c) \\ \sum_{\substack{(c_1',c_2',n',a',dir',max') \in T' \wedge Inclinación(c_1',c_2') \leq \alpha \wedge \\ (c_1',c_2',n',a',dir',max') \neq (c_1,c_2,n,a,dir,max)}} n' & si \quad (dir = 0 \Rightarrow c_2 = c) \wedge (dir = 1 \Rightarrow c_1 = c) \end{cases}$$

Para calcular i se considera que si 2 tramos tienen igual inclinación, primero se conecta al cruce el tramo que se dirige hacia c, y en el primer caso ($(dir = 1 \Rightarrow c_2 = c) \wedge (dir = 0 \Rightarrow c_1 = c)$) t tiene dirección hacia c. Entonces se suman la cantidad de carriles de los tramos con inclinación menor estricto que α, porque si hubiera otro tramo con igual inclinación seguro ocupa las posiciones posteriores a t.
En el segundo caso ($(dir = 0 \Rightarrow c_2 = c) \wedge (dir = 1 \Rightarrow c_1 = c)$) t tiene dirección opuesta a c, entonces se suman la cantidad de carriles de los tramos con inclinación menor o igual que α porque si hubiera otro tramo con igual inclinación seguro ocupa las posiciones anteriores a t.

Finalmente, las tuplas de V son separadas en aquellas que corresponden a tramos de ingreso al cruce y las que corresponden a los de salida. Esto se logra chequeando la dirección de circulación de los vehículos definida para cada tramo. Así se construye el conjunto In con los tramos de entrada a (c, maxc) y el conjunto Out con los de salida.

In = { (t,i) / (t, i,α) ∈ V ∧ t = (c1,c2, n, a, dir, max) ∧ [(c1 = c ∧ dir = 0) ∨
(c2 = c ∧ dir = 1)] }

Out = { (t,i) / (t, i,α) ∈ V ∧ t = (c1,c2, n, a, dir, max) ∧ [(c1 = c ∧ dir = 1) ∨
(c2 = c ∧ dir = 0)] }

devolver (In, Out)

Fin Ports_In_Out.

Esta función es idéntica a la definida en [DW99]

- Ctd_Celdas_Cruce (c: Cruces, T: Π(Tramos)): N

  Esta función establece el número de celdas que contendrá el cruce c; sumando la cantidad de carriles que contiene cada tramo acoplado. Pues en cada cruce existirá una celda por cada carril de cada tramo acoplado.

  Sea c = (p, maxc)

  $$k = \sum_{\substack{t \in T \wedge t=(c1,c2,n,a,dir) \\ \wedge (c1=p \vee c2=p)}} n$$

  devolver (k)

Fin Ctd_Celdas_Cruce

Esta función es idéntica a la definida en [DW99]

Sección VII: Referencias

Bibliografía

[CDL97] *"A Cellular Automata Model for Urban Traffic and its aplications to the city of Genova"*
B. CHOPARD, A. DUPUIS, P. LUTHI
Proceedings of Traffic and Granular Flow
1997.

[CLQ96] *"Cellular Automata Model of Car Traffic in two-dimensional street networks"*
B. CHOPARD, P. LUTHI, P. QUELOZ
J. Phys. A, vol 29, pp. 2325-2336, 1996.

[CQL95] *"Traffic Models of a 2D road network"*
B. CHOPARD, P. QUELOZ, P. LUTHI
Proceedings of the 3rd CM users' Meeting
October 1995
Parma.

[DW99] *Definición de un lenguaje de especificación para simulación de tráfico urbano siguiendo el paradigma Cell-Devs.*
Alejandra Davidson, Gabriel A. Wainer Tesis de Licenciatura
Departamento de Computación Facultad de Ciencias Exactas y Naturales Universidad de Buenos Aires

[Gia96] *"Introduction a la modelisation et a la simulation"*.
GIAMBIASI, N.
Material del curso de D.E.A.
Univerite d' Aix-Marseille III. 1996.

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

[RW99] *Implementación de Modelos Cell-Devs n-dimensionales*
Daniel Rodriguez, Gabriel A. Wainer Tesis de Licenciatura
Departamento de Computación Facultad de Ciencias Exactas y Naturales Universidad de Buenos Aires - 1999

[W96] *INFORME TÉCNICO: Introducción a la Simulación de Eventos Discretos*
Gabriel A. Wainer Report n.: 96-005
Departamento de Computación Facultad de Ciencias Exactas y Naturales Universidad de Buenos Aires

[Wai98] "*Contribución a la Modelización y Simulación de Sistemas de Eventos Discretos*".
WAINER, G.

Tesis de Doctorado. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Argentina; y Universite' d'Aix-Marseille III. Francia. 1998.

[WFG97a] *"An environment for simulation of cellular DEVS models."*
G.A. Wainer, C. Frydman, N. Giambiasi.
En Proceedings of the 1997 SCS European Multiconference of Computer Simulation.
Estambul, Turquía.

[WG97] *INFORME TÉCNICO: Specification, modeling and simulation of times Cell-DEVS spaces*
Gabriel A. Wainer, Norbert Giambiasi Report n.: 97-006
Departamento de Computación Facultad de Ciencias Exactas y Naturales Universidad de Buenos Aires

[WG98] *"Specification, modelling and simulation of timed Cell-DEVS models".*
G. WAINER, N. GIAMBIASI
Technical Report 97-007, Departamento de Computación, FCEN/UBA.
Submitted to publication. 1998.

[WG98b] *"N-dimensional Cell-DEVS".*
G. WAINER, N. GIAMBIASI
Technical Report TR-98-017, Departamento de Computación, FCEN/UBA.

[Y97] *A simulation laboratory for evaluation of dynamic traffic management systems*
Qi Yang
Massachusetts Institute of technology
Junio 1997

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

Bibliografía Adicional

[AHU/93] *Network Flows*

Ahuja R., Magnanti T. y Orlin J.

Prentice Hall (1993).

[AHU88] *Estructuras de Datos y Algoritmos*
Alfred V. AHO, John E. HopCroft, Jeffrey D. Ullman

1988

[BBW98] *CD++: Una herramienta de implementación de modelos Cell-Devs binarios.*
Amir Barylko, Jorge Beyoglonian, Gabriel Wainer
Report n: 98-006
Departamento de Computación
Facultad de Ciencias Exactas y Naturales.
Universidad de Buenos Aires.

[CMB93] *Use of high speed cellular automata machine to simulate road traffic*
Paul Cockshott, George McCaskill, Peter Barric
1993

[CWM94] *Paramics- Moving vehicles on the connection Machine*
Gordon Cameron, Brian Wylie, David McArthur
IEEE 1994

[ES97] *Microscopic simulation of urban traffic based on Cellular Automata*
J. Esser and M. Schreckenberg
International Journal of Modern Physics C, Vol 8, No. 5 (1997) 1025 – 1036

[GAR/79]*Computers and Intractability: A Guide to the Theory of NP-Completeness.*

Garey M. y Johnson D., W. Freeman and Co (1979).

[HEL/88] *"Dijkstra's two-tree shortest path algorithm"*.

Helgason R., Kennington J. y Stewart B.

Technical report, Department of Operations Research and Engineering Management,

Southern Methodist University, Dallas, USA (1988).

[JOH/76] *"Lower bounds for selection in X+Y and other multisets"*.

Johnson D. y Kashdan S.,

Report No. 183, Computer Science Department, Pennsylvania State University,

University Park, USA.

[LO97]
Urban Operations Reserch
Richard C.Larson, Amadeo R. Odoni
Massachusetts Institute of technology 1997-99.

[NWWS97] *Two-lane traffic rules for cellular automata: A systematic approach*
K. Nagel, D. Wolf, P. Wagner, P. M. Simon
Physical Review E,
1997

[SG98] *A Cellular Automaton Model for Bi-Directional Traffic*
P. M. Simon , H. A.Gutowitz
Physical Review E, vol 57:2, pág. 2441-2444.
1998

[Wol86] *Theory and Applications of Cellular Automata*.
WOLFRAM, S.
Vol. 1, Advances Series on Complex Systems.
World Scientific, Singapore, 1986.