UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingenería
Departamento de Ciencias de la Computación y TI

Análisis y diseño de protocolos de comunicación entre vehículos en un sistema de tránsito modelado como un sistema distribuido

Trabajo de graduación presentado por
Cristian Gustavo Castro Xum
para optar al grado académico de Licenciado en Ingeniería en Ciencias de la Computación y Tecnologías de la Información.

Guatemala
2016

Análisis y diseño de protocolos de comunicación entre vehículos en un sistema de tránsito modelado como un sistema distribuido

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Departamento de Ciencias de la Computación y TI

Análisis y diseño de protocolos de comunicación entre vehículos en un sistema de tránsito modelado como un sistema distribuido

Trabajo de graduación presentado por

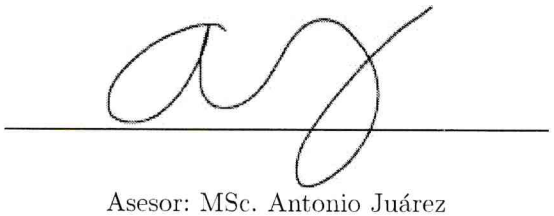
Cristian Gustavo Castro Xum

para optar al grado académico de Licenciado en Ingeniería en Ciencias de la Computación y Tecnologías de la Información.

Guatemala

2016

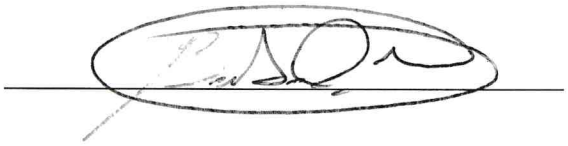
Vo.Bo.:



Asesor: MSc. Antonio Juárez

Tribunal Examinador:

Terna 1: Ing. Marta Ligia



Terna 2: Ing. Bidkar Pojoy

Terna 3: Ing. Luis Figueroa

Fecha de aprobación: Guatemala, 16 del mes de Noviembre del 2016 .

Mi eterno agradecimiento en primer lugar a Dios

quien me ha guiado durante mi camino.

Luego a mis padres quienes me apoyaron y se han esforzado por mí

Carmen Xum y Mateo Castro

Tercero a mis hermanos y hermanas:

Miriam Chali y Elida Chali

Aroldo Chali y Edy Chali

Por último, a mi mejor amiga:

Alejandra González

ÍNDICE

Lista de figuras

NOTACIÓN

Abreviaciones y siglas

i.e.	es decir
e.g	por ejemplo
BSM	Mensaje de seguridad básica.
CC-IP	Protocolo de Intersección de Cruce Concurrente
CDAI	Algoritmo de detección de colisiones para intersecciones.
CRS	Segmento de vía actual.
DSRC	Comunicación Dedicada a Rangos Cortos.
FCFS	Celda que intersecta una trayectoria de celdas
GPS	Sistema de Posicionamiento Global
MP-IP	Protocolo de Intersección de Máximo Progreso
NRS	Segmento de vía siguiente.
PROMELA	Lenguaje de modelado para verificación de procesos en sistemas distribuidos.
SPIN	Herramienta de verificación de software de código libre.
TCT	Tabla de trayectoria de celdas.
TCL	Lista de trayectoria de celdas.
TIC	Celda que intersecta una trayectoria de celdas
V2V	Comunicación Vehículo a Vehículo

Resumen

Azimir *et al.* en el trabajo de investigación *Manejo de Intersecciones utilizando Redes Vehiculares* propone el protocolo MP-IP, destinado para la comunicación entre vehículos autónomos en intersecciones; el objetivo de este protocolo es aumentar la seguridad y rendimiento en el tráfico. Los resultados de este protocolo muestran que el uso del protocolo aumenta el rendimiento del tráfico en un 83 % comparado con el sistema de luces tradicional. Sin embargo, es importante garantizar la confiabilidad de este protocolo para evitar posibles daños materiales o humanos en su implementación en un escenario real.

En el presente trabajo se aplica el método de *Verificación de modelos* al protocolo MP-IP con el fin de verificar que el diseño de MP-IP es correcto bajo un modelo y propiedades formales. Para modelar el diseño del protocolo se utilizó el lenguaje de verificación de modelos PROMELA, además, se planteó un conjunto de propiedades que el protocolo debe cumplir para verificar su correctitud; estas propiedades fueron plantedas formalmente utilizando Lógica Temporal Lineal. Las propiedades fueron verificadas por medio de la herramienta SPIN, adicionalmente se verificó la ausencia de bloqueo mutuo y código inalcanzable en el protocolo utilizando esta herramienta.

Finalmente, los resultados de SPIN mostraron que el protocolo cumple con las propiedades planteadas, es libre de bloqueo muto y código inalcanzable. Por lo tanto, el protocolo MP-IP es correcto bajo esa especificación.

1. Introducción

El presente trabajo tiene como objetivo verificar formalmente la correctitud del protocolo MP-IP. Este protocolo está destinado para la comunicación entre vehículos autónomos en escenarios de intersección. Para verificar la correctitud del protocolo se utilizó el método de verificación llamado *Verificación de Modelos*.

Principalmente, este trabajo se fundamenta a partir de tres temas: Sistemas Distribuidos, Verificación de Modelos y Comunicación Vehicular. En el Capítulo 4 se hace referencia a cada uno de estos temas haciendo énfasis en los elementos que se consideran importantes para el desarrollo de este trabajo. Además, se expone a detalle el protocolo MP-IP.

En el Capítulo 5, se expone la aplicación del método de Verificación de Modelos. Este método se aplicó desarrollando un modelo formal del protocolo MP-IP en PROMELA, asimismo, se planteó un total de siete de propiedades que garantizan la correctitud del protocolo. Cinco de estas propiedades fueron planteadas utilizando Lógica Temporal Lineal. Las propiedades restantes corresponden a la verificación de ausencia de código inalcanzable y bloqueo mutuo.

A partir del modelo y las propiedades en PROMELA, se utilizó la herramienta SPIN para verificar la correctitud del protocolo. Los resultados de esta herramienta, expuestos en el Capítulo 6, muestran que el protocolo cumple con todas las propiedades planteadas en Lógica Temporal Lineal. Adicionalmente, no se reportaron escenarios de bloqueo mutuo y código inalcanzable.

Finalmente, en el Capítulo 7, a partir de los resultados de SPIN se concluye que el protocolo MP-IP es correcto bajo la especificación planteada en este trabajo, además, se realiza un análisis de los resultados mostrados por el verificador de modelos SPIN y de las propiedades planteadas en el Capítulo 5.

2. Justificación

La explosión del primer Ariane 5 [2], el choque en Marte de La Mars Climate [19], la sobredosis de radiación en pacientes ocasionados por la Therac 25 [5] y el hundimiento del buque HMS Sheffield (D80) [16] son solamente algunos de los casos en los que errores críticos en el diseño de software han provocado que proyectos valuados en millones de dólares fracasen completamente o dejen pérdidas humanas.

Actualmente, uno de los campos activos de investigación es el diseño de protocolos de comunicación vehicular. Considerando las pérdidas materiales y humanas que han provocado errores en diseño de software de proyectos importantes, es vital asegurarse que los protocolos de comunicación vehicular sean confiables para su implementación en el mundo real, ya que un error en diseño podría costar la vida de muchas personas.

Específicamente, el protocolo MP-IP presenta un 83 % de mejora en el rendimiento del tráfico en intersecciones [2]. Tomando en cuenta este resultado, el protocolo MP-IP podría ser implementado en escenarios reales que involucren intersecciones con el fin de disminuir congestionamientos vehiculares. En el caso de ser implementado, será importante verificar la correctitud de este protocolo para excluir la posibilidad de que fallas del algoritmo causen daños materiales o humanos.

Para comprobar si el protocolo MP-IP es correcto se utilizó el método de verificación formal llamado *verificación de modelos*, ya que es un método riguroso que explora exhaustivamente todos los posibles comportamientos del protocolo. Asimismo, las propiedades del sistema fueron planteadas en Lógica Temporal Lineal dado que ésta permite plantear propiedades de sistemas concurrentes.

3. Objetivos

3.1. Objetivos generales

- Verificar formalmente la correctitud de un protocolo de comunicación para la coordinación vehicular en las intersecciones.

3.2. Objetivos específicos

- Plantear un conjunto de propiedades en Lógica Temporal Lineal que garanticen la correctitud del protocolo MP-IP.
- Desarrollar un modelo formal del protocolo MP-IP en PROMELA.
- Transcribir las propiedades de verificación en Lógica Temporal Lineal del protocolo MP-IP a PROMELA.
- Aplicar el método de Verificación de Modelos utilizando el modelo en PROMELA y las propiedades en Lógica Temporal Lineal por medio de la herramienta SPIN.
- Determinar si el protocolo MP-IP es correcto a partir de los resultados de SPIN.

4. Marco teórico

4.1. Sistemas distribuidos

No existe una definición universal sobre qué es exactamente un sistema distribuido (Silva, 2004); sin embargo, algunos autores han hecho sus propias definiciones:

"Un sistema distribuido consiste en una colección de computadoras autónomas enlazadas por una red y equipadas con un sistema de software distribuido"
(Tanembaum, 1995)

"Definimos un sistema distribuido como aquel en el que los componentes de hardware o software, localizados en computadoras unidas mediante red, comunican y coordinan sus acciones sólo mediante el paso de mensajes" (Coulouris, 2001)

A partir de las definiciones anteriores se derivan los siguientes elementos:

- Concurrencia: En una red de computadoras, la ejecución de programas es la norma. Cada uno puede realizar su trabajo en su computador compartiendo recursos cuando sea necesario (Silva, 2004).
- Inexistencia de reloj global: Cuando los programas necesitan cooperar, coordinan sus acciones mediante el intercambio de mensajes. La coordinación depende del instante en que se realicen las acciones de los programas o procesos; sin embargo, resulta que hay límites en la precisión de la sincronización de los relojes locales dconociendo a e cada una de las computadoras. Esto es una consecuencia directa del paso de mensajes a través de los nodos de una red (Silva, 2004).
- Fallos independientes: Cada componente del sistema puede fallar independientemente, sin afectar el funcionamiento de la red entera (Silva 2004).

FIGURA 4.1: La red mundial de computadoras es un ejemplo de sistema distribuido. Bouchrika (2013)

4.1.1. Comunicaciones en Sistemas Distribuidos Los mecanismos de variables compartidas cuyo acceso se sincroniza mediante cerrojos de exclusión mutua u otras primitivas (variables condición, cerrojos de lectores-escritores), han sido estudiados desde hace mucho tiempo. Lo mismo cabe decir sobre el paso de mensajes, que implementa colas FIFO sobre almacenamiento temporal en memoria, a menudo integrados en el sistema de ficheros (Silva, 2004)

En los sistemas débilmente acoplados (habitualmente redes), el paso de mensajes parece el mecanismo de comunicación natural. En un sistema distribuido, las necesidades de comunicación conducen a utilizar esquemas específicos de gestión de los recursos para los que el paso de mensajes resulta adecuado (Silva, 2004)

4.1.2. Comunicaciones basada en paso de mensajes El paso de mensajes comprende un conjunto de mecanismos que permiten comunicar procesos mediante un enlace o canal de comunicación, identificando el proceso origen o destino respectivamente en las primitivas de recibir o enviar. (Lafuente, 2011)

El modelo de programación del paso de mensajes para comunicación entre un sistema de varios nodos no difiere del de paso de mensajes para comunicación dentro de un nodo. El hecho de tener que confiar en los protocolos de red introduce dependencias derivadas del rendimiento y la fiabilidad del canal de comunicación entre nodos. (Lafuente, 2011)

Las características principales asociadas a un canal de comunicación son las siguientes:

- **Modo de sincronización** Por una parte, si el canal de comunicación está ocupado, la primitiva de enviar puede bloquear al proceso hasta que se libere el canal (lo que depende también de si permite almacenamiento temporal o no) y pueda depositar el mensaje (modo bloqueante o síncrono). La alternativa (modo no bloqueante) permite que el proceso continúe aunque el mensaje no se haya podido enviar, transfiriendo a la aplicación la responsabilidad de gestionar la sincronización en el uso del almacenamiento temporal de usuario donde se ubica el mensaje enviado. (Lafuente, 2011)
- **Fiabilidad** En lo referente a la fiabilidad de la comunicación, el mecanismo de paso de mensajes depende del soporte que le proporcione la red. Si se implementa sobre un protocolo de transporte seguro, la comunicación se considera fiable, en el sentido de que el emisor puede confiar en que el receptor acabe por recibir el mensaje correctamente o sea informado de lo contrario. Esto es a costa de una cierta sobrecarga por la necesidad que tiene el protocolo de confirmar las recepciones. En cambio, un mecanismo no fiable permitirá una comunicación menos costosa, pero delega en la aplicación la responsabilidad de verificar la corrección de la comunicación. (Lafuente, 2011)
- **Modo de comunicación** En sistemas distribuidos es de gran interés el soporte de primitivas de paso de mensajes de 1:N. La difusión ancha permite enviar un mensaje a todas las direcciones accesibles por el emisor y se usa en redes locales. Un caso particular de difusión amplia es el de multidifusión, el cual permite seleccionar un subconjunto de direcciones a las que enviar el mensaje. El soporte para multidifusión es muy útil en sistemas replicados, como se verá más adelante. Como ejemplo, el protocolo IP reserva un conjunto de direcciones para multidifusión. (Lafuente, 2011)

4.1.3. Tecnologías desarrolladas para Sistemas Distribuidos Durante los últimos años el mundo de la computación gira en torno a los sistemas distribuidos que involucran dispositivos conectados a una red que formen una comunidad de intercambio de información y servicios. De esto, han surgido tecnologías cuyo objetivo fundamental es la interoperabilidad entre los dispositivos de red. A continuación se describen algunas de ellas.

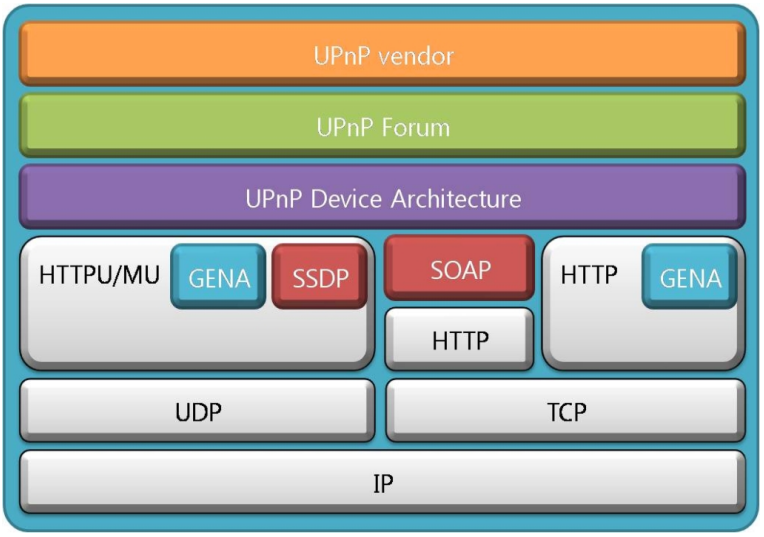
- **Universal Plug and Play (UPnP)** UPnP (Universal Plug and Play) es una arquitectura destinada para la conexión en red de aplicaciones inteligentes, dispositivos inalámbricos y ordenadores personales de cualquier tipo. Está diseñada para proporcionar unos estándares de conectividad flexibles y fáciles de usar a redes ad-hoc, ya sea en el hogar, en un pequeño negocio, en espacios públicos con o sin conexión a Internet. UPnP es una arquitectura de red abierta y distribuida

que incorpora TCP/IP [1] y tecnologías web para permitir el trabajo en red además de tareas de control y transmisión de datos entre los dispositivos de la red en los entornos comentados. (Gómez, 2012)

La arquitectura UPnP contiene componentes de Internet, incluyendo IP [2], TCP [3], UDP [4], HTTP [5] y XML[6] . Como en Internet, la comunicacion está basada en protocolos declarativos, expresados en XML y comunicados via HTTP. La apuesta de UPnP por IP está fundamentada en su capacidad para funcionar en distintos medios físicos, para permitir la inter-operacion entre distintos fabricantes y así alcanzar sinergias con Internet y la mayoría de las intranets domésticas y de oficina. UPnP ha sido explícitamente diseñado para acomodarse a esos entornos. (Gómez, 2012)

A continuación se muestra la arquitectura de la pila de protocolos de UPnP.

FIGURA 4.2: Pila de protocolos de UPnP. Huerta et al. (2005)



En las capas superiores, los mensajes sólo contienen información específica del fabricante, acerca de sus dispositivos. Más abajo de la pila, el contenido del fabricante es complementado con información definida por el UPnP Forum. Los mensajes de las capas superiores son utilizados en protocolos específicos como el Protocolo Simple de Descubrimiento de Servicios (SSDP, Simple Service Discovery Protocol) y la Arquitectura General de Notificación de Eventos (GENA, General Event Notification Architecture), entre otros. Estos mensajes son enviados mediante HTTP, bien en modo multidifusión o unidifusión sobre IP. (Gómez, 2012)

[1] Protocolo de Control de Transmisión/Protocolo de Internet
[2] Protocolo de Internet
[3] Protocolo de Control de Transmisión
[4] Protocolo de Datagrama de Usuario
[5] Protocolo de Transferencia de Hipertexto
[6] Lenguaje de Marcas Extensible

- **Aplicaciones de posicionamiento Global en Sistemas Distribuidos**
  Los Sistemas de Posicionamiento Global (GPS) permiten la determinación precisa de la ubicación, velocidad, dirección y el tiempo. GPS es un sistema que tiene como objetivo la determinación de coordenadas espaciales de puntos en un sistema de referencia mundial. Los puntos pueden estar ubicados en cualquier lugar del planeta,pueden permanecer estáticos o en movimiento y las observaciones pueden realizarse en cualquier momento del día (Huerta *et al.*, 2005).
  Para la obtención de coordenadas, el sistema se basa en la determinación simultánea de las distancias a cuatro satélites (como mínimo) de coordenadas conocidas. Estas distancias se obtienen a partir de las señales emitidas por los satélites, las que son recibidas por receptores especialmente diseñados. Las coordenadas de los satélites son provistas al receptor por el sistema. (Huerta *et al.*, 2005).
  El sistema GPS está siendo aplicado muy a menudo en sistemas distribuidos. Muchos de los esfuerzos se centran en proveer una mejor guía de navegación a conductores por medio de GPS. (Dommety, 1998)
  Algunas de las aplicaciones se incluyen a continuación: conmutación de circuitos usando relojes sincronizados, sincronización del reloj en sistemas distribuidos, sincronización de bases de datos, comunicación en tiempo real sin conexión, localización de recursos, protocolos de adaptación ubicación,antenas direccionales, comunicaciones vehiculares, entrega de correo electrónico basado en la ubicación geográfica, el control de robots distribuido, y por último la navegación y ubicación de los equipos marcado por el personal de mantenimiento (Dommety, 1998)

4.2. Verificación de protocolos y sistemas de información

4.2.1. Confiabilidad de los sistemas de información Tecnologías de la información y comunicación están creciendo rápidamente, estos sistemas se han vuelto más y más complejos, a su vez nos invaden por medio del internet en cualquier tipo de sistemas embebidos como tarjetas inteligentes, computadoras de mano, teléfonos móviles, entre otros. Sin embargo, es importante hacer un énfasis especial en la confiabilidad de los sistemas de información, ya que históricamente fallas ocurridos en software han dejado daños irreversibles. Tal es el caso de algunos sistemas como el control de software del misil Ariane-5, el explorador Mars y los aviones de la familia *Airbus*, los cuales llenaron de títulares los periódicos de todo el mundo. Así pues, sistemas de software similares al de estos sistemas son utilizados por plantas nucleares, sistemas de alerta y algunos otros como plantas químicas. Claramente, errores en software en estos sistemas pueden provocar consecuencias desastrosas. (Baier *et al.*, 2008)

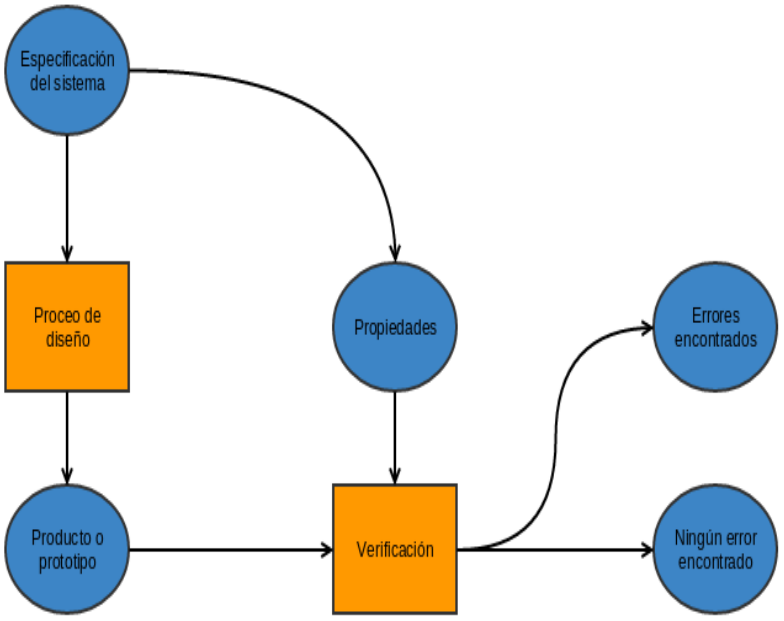
FIGURA 4.3: El despegue del misil Ariane-5 el 4 de Junio de 1996 explotó 36 segundos después del lanzamiento debido a un error de conversión de 64 bits de punto flotante a 16 bits de valor entero. Arnold (2000)

La creciente dependencia en aplicaciones críticas de sistemas de información llevan a afirmar lo siguiente

"La confiabilidad sobre los sistemas de las tecnologías de la información es un factor clave en el proceso de diseño del sistema." (Baier *et al.*, 2008)

4.2.2. Verificación de software Actualmente, diversas técnicas de verificación están siendo aplicadas en el diseño de Sistemas y Tecnologías de la Información con el fin de construir sistemas confiables. En pocas palabras, la verificación del sistema se utiliza para establecer que el diseño o producto bajo consideración posee ciertas propiedades. Las propiedades a ser validadas pueden ser muy elementales, como que el sistema nunca debe ser capaz de alcanzar alguna situación en la cual no se pueda continuar o retroceder (escenario de bloqueo mutuo); estas propiedades son obtenidas en la especificación del sistema. La especificación del sistema preescribe lo que el sistema es y no es capaz de realizar y a su vez constituye las actividades de verificación. A partir de esto, un defecto es encontrado cuando el sistema no cumple con las propiedades de la especificación del mismo. El sistema se considera "correcto" cuando satisface todas las propiedades obtenidas de su especificación. De esto, se concluye que la correctitud del sistema está relacionada con su especificación y no se trata de una propiedad del sistema. (Baier *et al.*, 2008) Una esquemática vista de verificación se muestra a continuación

FIGURA 4.4: Proceso de verificación de sistemas. (Baier *et al.*, 2008)



A partir de acá, se trabajará con una técnica llamada verificación de modelos, la cual inicia a partir de una especificación formal del sistema.

4.2.3. Verificación de modelos En el diseño de software y hardware complejo, se emplea más tiempo y esfuerzo en la verificación que en la construcción. Los métodos formales ofrecen un gran potencial para obtener una rápida integración de la verificación del sistema en el proceso del diseño. (Baier *et al.*, 2008)

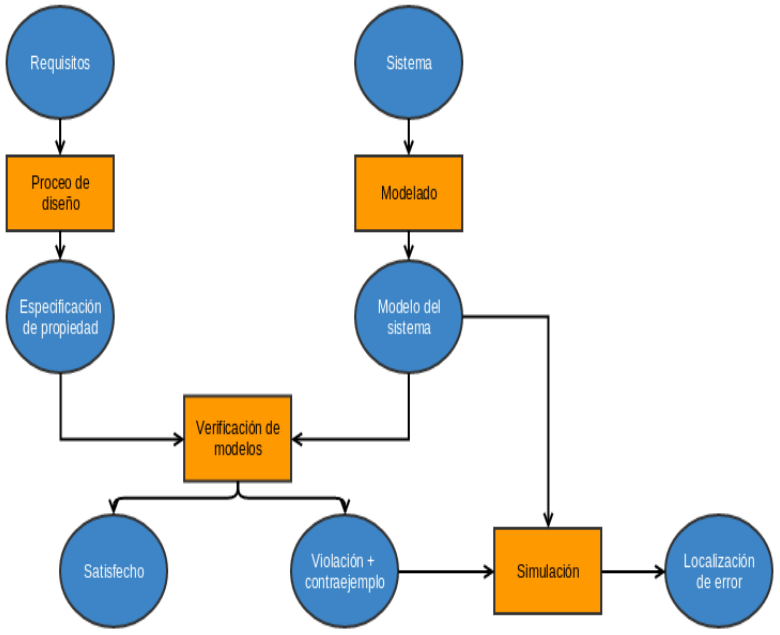
Discutiendo brevemente acerca del rol de los métodos formales en la verificación de sistemas, iniciamos mencionando que los métodos formales son considerados como "las matemáticas aplicadas para modelar y analizar Sistemas y Tecnologías de la Información". Su objetivo es establecer la correctitud del sistema con rigor matemático. Su gran potencial ha causado que los ingenieros aumenten el uso de métodos formales para la verificación de sistemas de software y hardware complejos. Además, los métodos formales, son una de técnicas de verificación altamente recomendadas para el desarrollo de software, por ejemplo, las prácticas utilizadas por IEC(Comisión Electrónica Internacional), los estándares de ESA (Agencia Espacial Europea) y los reportes de investigación de FAA (Autoridad Federal de Aviación) y NASA (Aeronautica Nacial y Administración Espacial) acerca de los métodos formales concluyen que (Baier *et al.*, 2008):

"Métodos formales deberían ser parte de la educación de todo Científico de la Computación e Ingeniero de Software, como una rama de las matemáticas aplicadas es una parte esencial de la educación de los ingenieros." (Baier *et al.*, 2008)

Las técnicas de verificación basadas en modelos describen el posible comportamiento del sistema en un lenguaje matemático y no ambiguo. Suele pasar que al obtener el modelo exacto del sistema, se descubren incompletitudes, ambigüedades e inconsistenas en las especificaciones informales del sistema, tales errores suelen ser encontrados posteriormente a la etapa de diseño. Esto provee la base para una gran variedad de técnicas de verificación desde una exhaustiva exploración (verificación de modelos) hasta experimentos con escenarios reestrictivos en el modelo (simulacion) o bien, en la realidad (pruebas). Sin embargo, cabe mencionar que, cualquier método de verificación basado en modelos, únicamente puede ser tan bueno como el modelo del sistema. (Baier *et al.*, 2008)

La verificación de modelos es una técnica que explora todos los posibles estados en una manera de fuerza bruta. De esta manera, se puede demostrar que un sistema satisface correctamente una propiedad. Se ha visto que los mejores verificadores de modelos pueden soportar espacios de estados de hasta 10^8 y 10^9 estados con explícita enumeración de espacios de estado. Utilizando algoritmos inteligentes y estructuras de datos adaptadas se puede llegar a utilizar espacios de estado mucho más grandes (10^{20} hasta 10^{476}) para problemas muy específicos. (Baier *et al.*, 2008)

FIGURA 4.5: Vista esquemática del enfoque de verificación de modelos



Baier *et al.*, 2008

Propiedades típicas pueden ser chequeadas utilizando verificación de modelos de una forma cualitativa natural, por ejemplo: ¿Se generó correctamente el resultado?, ¿Puede el sistema alcanzar una situación de bloqueo mutuo cuando dos sistemas concurrentes están esperando el uno al otro y esto provoca la finalización del sistema? Pero además, propiedades del tiempo pueden ser verificadas, ¿Puede ocurrir un bloqueo mutuo una hora después de que el sistema se reinicie?, ¿Se recibe una respuesta siempre cada 8 minutos? La verificación de modelos requiere una precisa y unambigua declaración de propiedades a ser examinadas. (Baier *et al.*, 2008)

La verificación de modelos ha sido aplicado exitosamente en varios sistemas de las tecnologías de la información. Por ejemplo, casos de bloqueo mutuo han sido detectados en sistemas de líneas aereas para reservaciones en línea, protocolos recientes de comercio electrónico también han sido verificados mediante esta metodología y varios estudios de los estándares internacionales de la IEEE para protocolos de comunicación domésticos han dejado mejoras notables en las especificaciones del sistema. (Baier *et al.*, 2008)

4.2.4. Características de la verificación de modelos Las siguientes secciones se enfocan en los detalles de las técnicas elementales para la verificación de modelos.

4.2.4.1. Proceso de verificación de modelos

- *Fase de modelación*
  - Modelar el sistema bajo la descripción del modelo
  - Formalizar las propiedades para ser verificadas.
- *Fase de ejecución* Ejecutar el verificador de modelos para verificar la validez de las propiedades en el modelo del sistema.
- *Fase de análisis* Analizar propiedades satisfechas o violadas si es que las propiedades no satisfacieron las condiciones requeridas.

4.2.5. SPIN Spin fue desarrollado por Gerald J. Holzmann en Laboratorios Bell en los años de 1980 con el fin de verificar varios protocolos de telecomunicaciones. Spin es una herramienta para la verificación de modelos y es utilizada frecuentemente para analizar la consistencia lógica de sistemas distribuidos, específicamente para los protocolos de comunicación. La primera versión de Spin-tool fue liberada en 1989 y desde entonces ha estado en constante desarrollo. En Abril de 2002 la herramienta fue premiada por la ACM (Asociación de Sistemas Informáticos) (Løvengreen, 2006).

SPIN ha sido utilizado en muchos sistemas críticos, incluyendo una gran variedad de misiones de la NASA. Para mayo información de la herramienta, es posible encontrar cocumentación y material de referencia en la página oficial de Spin, la cual es de código libre y es gratis para propósitos educacionales (Løvengreen, 2006).

http://spinroot.com

Spin puede ser trabajado como un simulador y un motor de verificación. El sistema es descrito en un lenguaje de modelación llamado **PROMELA**. Este lenguaje permite la creación dinámica de sistemas concurrentes. Comunicación vía mensajes de canales puede ser definida de forma asíncrona o síncrona (Ben-Ari, 2008).

Dado un modelo en PROMELA, SPIN puede realizar simulaciones interactivas o aleatorias de la ejecución del sistema o puede generar un programa en lenguaje C que realiza una exhaustiva verificación del espacio de estados. Durante simulaciones y verificaciones SPIN chequea por la ausencia de bloqueos mutuos, recepciones inespecificadas o código inalcanzable. El verificador también puede proveer correctitud sobre las invariantes de un sistema y encontrar ciclos en los cuales no se pueda progresar (*non progress cycles*) (Ben-Ari, 2008)

4.2.5.1. ¿Cómo trabaja SPIN? Usualmente los verificadores construyen internamente una representación del modelo para ser analizada y luego utiliza esto para generar una representación del espacio de estados alcanzable. Por razones de eficiencia, SPIN genera un *programa de verificación* para cada tarea de verificación. Esto

es un programa en ANSI C llamado **pan** junto a un número de archivos auxiliares. La tarea de verificación puede ser lograda compilando y ejecutando el programa pan (con varias opciones) (Løvengreen, 2006).

4.2.5.2. Verificación de algoritmos y opciones en SPIN Básicamente SPIN realiza un trabajo de verificación construyendo un conjunto de estados alcanzables a partir de la interacción de acciones. Para saber si un estado ya fue utilizado se emplea una tabla hash. Además, se aplican un conjunto de técnicas destinadas para la compresión de estados. Usualmente los estados son visitados utilizando el algoritmo de búsqueda en profundidad, pero también puede ser elegido el algoritmo de búsqueda en anchura (Løvengreen, 2006).

Para sistemas más grandes, una búsqueda exhaustiva podría no ser factible debido a limitantes de espacio. En este caso el modo *bitstate* podría ser seleccionado, ya que con esta técnica los estados visitados no se almacenan en memoria sino que solamente se marcan sus códigos hash. (Løvengreen, 2006).

Normalmente el algoritmo de *Reducción parcial de orden* se utiliza por defecto. Este algoritmo es utilizado para reducir el espacio de estados que se usará para realizar la búsqueda (Løvengreen, 2006).

SPIN provee una herramienta de reporte sobre afirmaciones que nunca se cumplen, llamada *never claims*. *Never claims* se utiliza para definir el compartamiento del sistema que por alguna razón es de especial interés. Es comunmente utilizado para especificar un comportamiento que *nunca* debe ocurrir. Las afirmaciones (*assertions*) son definidas como una serie de proposiciones o expresiones booleanas en los que el sistema de estados debe satisfacer la secuencia de estados especificada (Løvengreen, 2006).

4.2.5.3. Búsqueda de ausencia de bloqueo mutuo en SPIN Un tipo especial de propiedad son los *bloqueos mutuos*. En general, un grupo de procesos están en *bloqueo mutuo* cuando todos ellos están bloqueados, esperando por alguna condición que debe ser establecida por otro proceso en el grupo (Løvengreen, 2006).

SPIN no es capaz de analizar qué proceso se encuentran esperando un conjunto de procesos bloqueados. De cualquier manera, si una situación ocurre en la que **todos** los procesos activos se encuentran bloqueados (o terminados), SPIN reportará una situación de bloqueo mutuo (Løvengreen, 2006).

Para evitar que procesos que efectivamente tienen un fin establecido sean reportados como bloqueo mutuo, para ello se declara un *localidad de control* en el código como un punto final aceptable. Por defecto, las *localidades de control* finales de cada proceso son considerados como estados finales aceptables (Løvengreen, 2006).

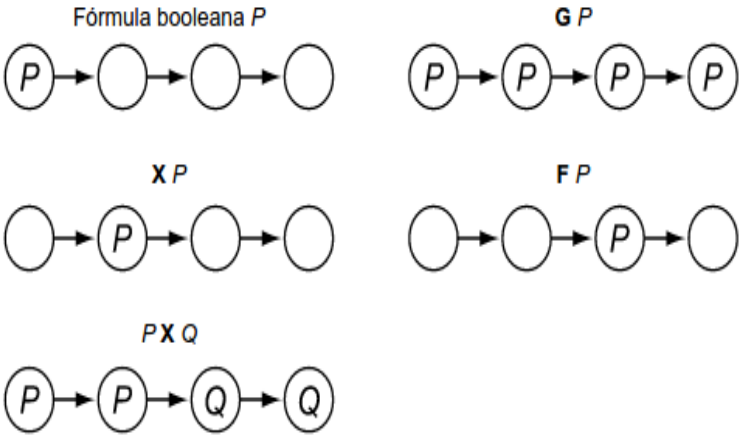
4.2.6. Lógica temporal lineal La lógica temporal se encuentra dentro de las llamadas lógicas temporales que se utilizan para modelar el tiempo y expresar propiedades del mismo, donde el tiempo es modelado como una secuencia infinita de estados. Existen diferentes formalismos de lógica temporal; tales como Lógica Temporal Lineal (LTL) y de Árbol Lógico Computacional (CTL). Específicamente, la LTL modela el tiempo como una secuencia lineal de estados. Los operadores temporales describen propiedades acerca del comportamiento de los estados del sistema a través del tiempo. (Garis, 2010) A continuación se muestran algunos de los operadores temporales:

NOTACIÓN 4.1: Operadores LTL

Operador	**Descripción**
$\circ$	El operador unario $\circ$ se utiliza para denotar que una proposición P, ocurrirá en el **próximo** momento del estado actual de P. Ocasionalmente el operador $\circ$ se denota por X.
$\diamond$	. El operador unario $\diamond$ se utiliza para denotar que la P ocurrirá en **algún** estado de la secuencia lineal de estados. Ocasionalmente el operador $\diamond$ se denota por F
$\square$	El operador unario $\circ$ es utilizado para afirmar que P ocurre en **todos** los estados de la secuencia lineal de estados.
U	El operador binario U se utilizado para afirmar que la prosición P es verdadera hasta que la propiedad Q lo sea. Esto en la secuencia lineal de estados.

A continuación se muestra una gráfica con la secuencia lineal de estados con cadda uno de los operadores anteriormente descritos.

FIGURA 4.6: Secuencia de estados en LTL. El operador X equivale a $\circ$, el operador F equivale a $\diamond$ y el operador G equivale a $\square$. Garis (2010)



Los operadores temporales son utilizados en conjunto con los operadores lógicos tradicionales (not, and, or), para definir las propiedades deseadas del sistema. Por ejemplo, propiedades de seguridad, en donde se establece que no es posible alcanzar un estado peligroso. Una fórmula descrita por medio de LTL se representa como una computación sobre una secuencia infinita de estados. (Garis, 2010)

4.3. Comunicación vehicular

4.3.1. Vehículos autónomos Los avances tecnológicos cada vez estrechan la distancia entre vehículos totalmente conducidos por humanos y vehículos automatizados (VA), estos últimos se refieren a vehículos capaces de ser conducidos por sí mismos total o parcialmente; es decir, los VA pueden no requerir de la presencia de un conductor humano. Estas tecnologías permiten al vehículo asistir y tomar decisiones que ayudan a los conductores humanos. Tales tecnologías incluyen Sistemas de Advertencia de Accidentes, Control de Velocidad Inteligente (ACC por sus siglas en inglés), sistemas para el control de carril y tecnología para parqueo autónomo. (Anderson *et al.*, 2014)

National Highway Traffic Safety Administration(NHTSA) creó cinco niveles de jerarquía para categorizar los distintos niveles de automatización. A continuación detallamos cada uno de ellos:

- Nivel 0 (Ningún nivel de automatización): El conductor tiene siempre el control completo y único de las funciones primarias del vehículo (freno, direccionamiento, aceleración y fuerza motriz). Ejemplos de estos vehículos son los que proveen controles primarios como luces de emergencias, parabrisas, luz de giro, etc. (Anderson *et al.*, 2014)
- Nivel 1 (Algunas funciones específicas de automatización) : Automatización a este nivel involucra una o más funciones específicas de control automático del vehículo. El conductor tiene el control general de todo pero puede ceder autoridad limitada al vehículo de control primario o por ejemplo asistentes de ayuda (Asistentes de frenado dinámico en situaciones de emergencia). El vehículo puede tener múltiples capacidades que combinan ayuda individual del conductor y tecnologías de prevención de accidentes; sin embargo, no sustituye la vigilancia del conductor y tampoco asume la responsabilidad de conducir del conductor. (Anderson *et al.*, 2014)
- Nivel 2 (Funciones de automatización combinadas) : Este nivel implica la automatización de al menos dos funciones de control primario diseñadas para trabajar al unísono que alivian al conductor del control de esas funciones. La principal distinción entre el nivel 1 y nivel 2, es que en el nivel 2 las condiciones específicas de funcionamiento para el que está diseñado el sistema varían. Además, posee

un modo de funcionamiento automático que al activarse el conductor se desacopla físicamente del control, principalmente del volante y pedal al mismo tiempo. (Anderson *et al.*, 2014)

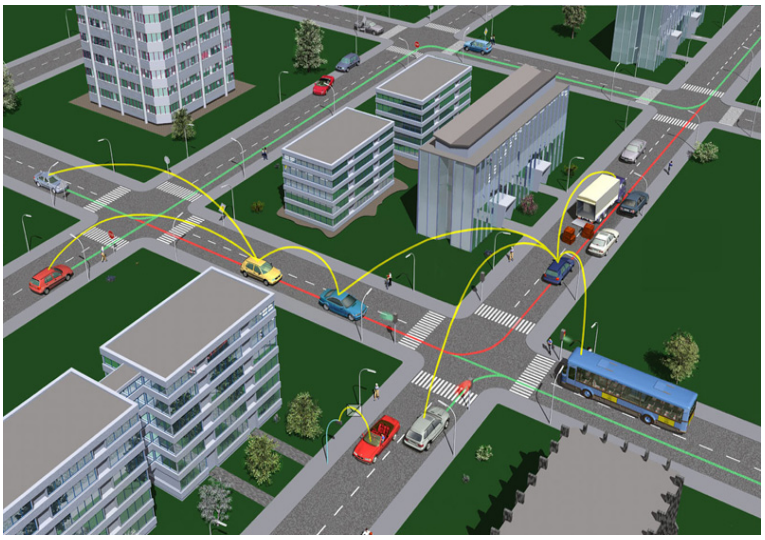
- Nivel 3 (Automatización limitada de auto-conducción): Vehículos en este nivel, involucra funciones de control del vehículo en situaciones críticas del tráfico. También se incluyen condiciones que dependen en gran medida que el vehículo sea capaz de monitorear cuando sea necesaria la transición de vuelta al control del conductor. Se espera que el conductor esté disponible para el control ocasional pero con suficiente tiempo de transición.
  El vehículo está diseñado para garantizar funcionamiento seguro durante el modo automatizado de conducción. Un ejemplo sería un sistema automatizado o de auto-conducción de automóvil que pueda determinar cuando el sistema ya no es capaz de soportar la automatización y el conductor deba volver a ejercer el control del vehículo. En escenarios de dirección contraria o áreas de construcción, el vehículo debe notificar al conductor que vuelva a participar en la tarea de conducir. La principal distinción entre los niveles 2 y 3 es que en el nivel 3, el vehículo está diseñado para que el conductor no tenga un seguimiento constante del camino mientras el auto se conduce. (Anderson *et al.*, 2014)
- Nivel 4 (Automatización de auto-conducción completa): El vehículo está diseñado para realizar todas las funciones de seguridad críticas de conducción y monitoreo del camino para un viaje completo. Tal diseño preveé que el conductor proporcionará destino o entrada de navegación, pero no se espera que esté disponible para el control del vehículo en cualquier momento durante el viaje. (Anderson *et al.*, 2014)

4.3.2. Comunicaciones dedicadas a rangos cortos La banda de red inalámbrica en Comunicaciones Dedicadas a Rangos Cortos (DSRC) es una banda de frecuencia reservada a aplicaciones seguras utilizando comunicación de Vehículo a Vehículo (V2V) y de Vehículo a Infraestructura (V2I). La tecnología DSRC ha capturado el interés de gobiernos, manufactureras automovilísticas, academia y otros actores que ofrecen latencia, exactitud, confiabilidad necesaria para seguridad activa y características de movilización en carreteras. Hasta la fecha varios protocolos de comunicación DSRC han sido propuestos y probados para demostrar su confiabilidad. Algunas aplicaciones de la tecnología DSRC han sido desarrolladas para la adquisición de datos, diseminación de sistemas y sistemas de advertencia en intersección. El radio de comunicación establecido por la DSRC es de 300 metros, dependiendo del ambiente. Las comunicaciones ocurren en una banda de 75 MHz del espectro de los 5.9 GHz (Li, 2010).

4.3.3. V2V (Comunicación vehículo a vehículo) V2V es la tecnología utilizada para evitar colisiones vehiculares, esta tecnología utiliza información enviada por vehículos cercanos para prevenir a un vehículo de situaciones peligrosas; por ejemplo, por medio de V2V se podría ayudar a un conductor de un vehículo a indicarle que debe reducir su velocidad o bien en un caso de intersección para saber si debe cruzar o no. Dicho intercambio de datos se realiza por medio de red inalámbrica. (Harding *et al.*, 2014). La comunicación V2V puede ser de dos tipos: entre dos vehículos y entre más de dos vehículos comunicándose en un mismo ambiente de red. El segundo tipo de comunicación V2V es usualmente utilizado para la adquisición de datos y diseminación de sistemas, además, facilita el uso de aplicaciones críticas con retardos de tiempo mínimo en redes ad hoc. (Harding *et al.*, 2014).

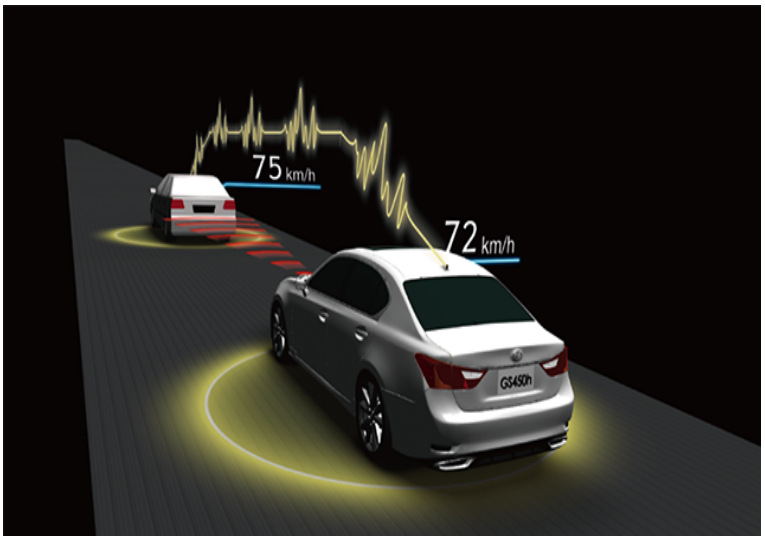
Para adaptar redes ad hoc a la comunicación vehicular, investigaciones extensas de redes han sido realizadas debido a la necesidad de comunicación rápida y diseminación de datos. (Harding *et al.*, 2014).

FIGURA 4.7: Distintos vehículos se comunican entre sí en medio del tráfico. Howard (2014)



4.3.4. ¿Cómo trabaja la comunicación V2V? Los sistemas de comunicación V2V están compuestos de dispositivos instalados en los vehículos que utilizan DSRC para intercambiar mensajes que contienen información vehicular, por ejemplo: velocidad, estado actual, histórico de recorrido. Los dispositivos de comunicación V2V usan información de otros vehículos y determinan si se debe advertir al conductor acerca de una posible situación de emergencia y así prevenir colisiones (Harding *et al.*, 2014).

FIGURA 4.8: Cada vehículo le envía información importante a otros como por ejemplo la velocidad. Toyota (2014)



4.3.5. Enfoques de investigación Algunos enfoques de investigación en comunicación vehicular actual son los siguientes:

- Identificación de escenarios críticos de colisión: este enfoque está muy avanzado y actualmente se trabaja en desarrollar aplicaciones para solucionar estos problemas enfocándose en rendimiento y efectividad de los mismos (Harding *et al.*, 2014).
- Asegurar interoperabilidad [7] y determinar la infraestructura necesaria para el desarrollo la comunicación V2V. Aplicaciones seguras deben funcionar en cualquier tipo de vehículo y adherir los estándares de comunicación para asegurar la integridad y seguridad del mensaje (Harding *et al.*, 2014).
- Desarrollar rigurosas métricas de seguridad. Desarrollo de medidores de rendimiento y pruebas efectivas de medición (Harding *et al.*, 2014).
- Desarrollar un prototipo con aplicaciones seguras y evaluar su desempeño. El desarrollo de estas aplicaciones ayuda al análisis de los requerimientos funcionales y rendimiento para tecnologías subyacentes tal como posicionamiento y comunicación. De cualquier manera, investigación adicional necesita ser llevada a casos más complejos como la búsqueda de escenarios de colisión difíciles de identificar. (Harding *et al.*, 2014).
- Desarrollar interfaces efectivas de vehículos que brinden confiabilidad en los sistemas de advertencias de colisiones, las cuales pueden afectar al rendimiento del conductor. (Harding *et al.*, 2014).

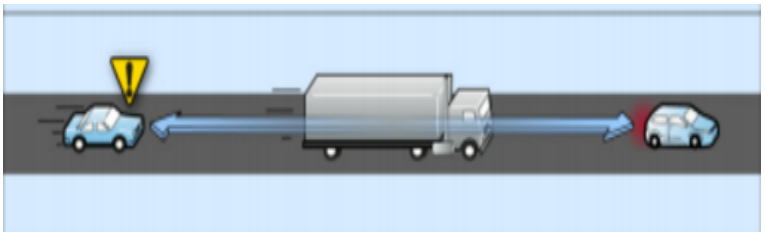
[7] El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) define interoperabilidad como la capacidad de dos sistemas de intercambiar información y utilizar dicha información para sus propias acciones

- Investigar problemas de políticas y formular decisiones regulatorias dentro de este contexto (Harding *et al.*, 2014).
- Desarrollar y evaluar comunicaciones seguras enfocadas a necesidades únicas y a la dinámica de los vehículos comerciales, furgones, tractores, motores. Investigaciones del NHTSA (National Highway Traffic Safety Administration) estiman que las aplicaciones de la comunicación V2V pueden reducir en un porcentaje significativo las colisiones de vehículos pesados (Harding *et al.*, 2014).
- Desarrollar aplicaciones de tránsito seguras. Utilizando trabajo ya realizado en seguridad automovilística y de tránsito puede impactar positivamente en la industria actual (Harding *et al.*, 2014).

4.3.6. Aplicaciones Algunos ejemplos de aplicaciones de comunicación vehicular en desarrollo son las siguientes (Harding (Harding *et al.*, 2014)., 2014) :

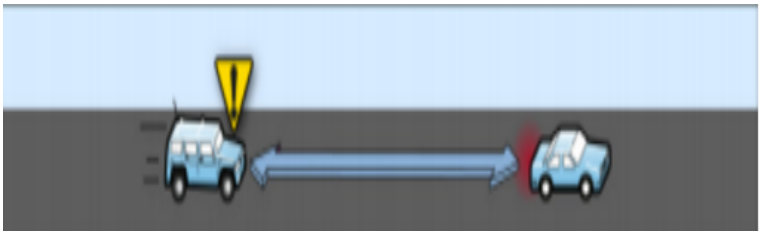
- Advertencia de luz de freno al conductor

FIGURA 4.9: Ejemplo de advertencia de luz de freno. (Harding *et al.*, 2014)



- Advertencia de colisión hacia adelante:

FIGURA 4.10: Ejemplo de advertencia de colisión al frente del vehículo. (Harding *et al.*, 2014)

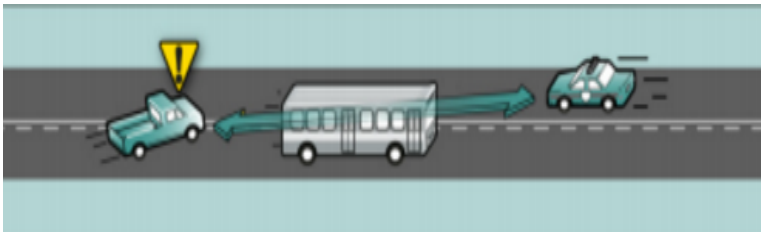


- Punto ciego: alerta en cambios de carril

FIGURA 4.11: Ejemplo de advertencia de cambio de carril. (Harding *et al.*, 2014)

- Advertencias de no rebasar

FIGURA 4.12: Ejemplo de advertencia para no rebasar. (Harding *et al.*, 2014)



- Advertencias de intersección ciega

FIGURA 4.13: Ejemplo de advertencia en intersección ciega. (Harding *et al.*, 2014)

Instituciones como la (NHTSA) han realizado investigación en V2V por más de una década en conjunto con otras instituciones de los Estados Unidos como el Departamento de Transporte (DOT), industria automotriz e instituciones académicas. En 2014 la NTHSA emitió noticias de reglamentación y un reporte del proyecto : "Comunicaciones Vehículo a Vehículo: Disponibilidad de tecnologías V2V para aplicaciones", la cual explora políticas legales y técnicas asociadas con comunicación V2V (Harding *et al.*, 2014).

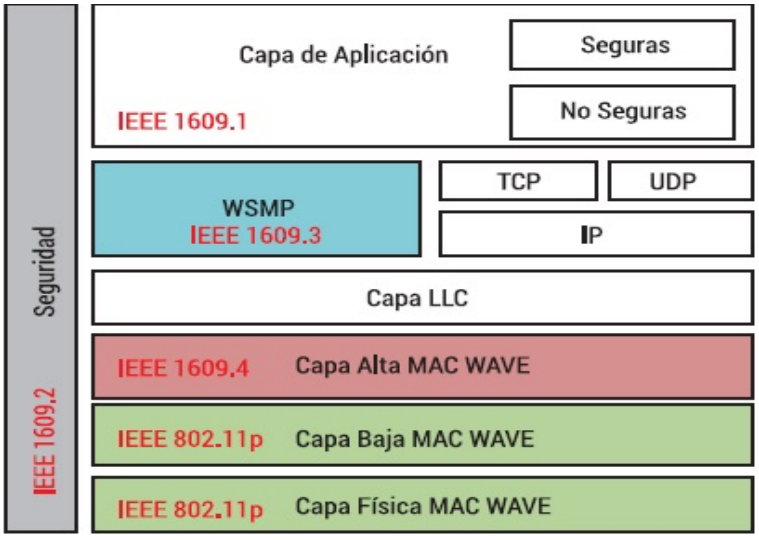
4.3.7. Estándares de comunicación Es importante asegurar la alta confiabilidad y bajo retardo en la construcción de sistemas de comunicación inalámbrica; ya que aunque la información sea entregada correctamente, si es entregada fuera del tiempo esperado, esta información no solamente ya no es útil sino también podría tener severas

consecuencias para la seguridad del tráfico. En la mayoría de los casos, las arquitecturas de redes ad hoc son la base de comunicación directa vehículo a vehículo. El estándar IEEE 802.11p está diseñado para comunicación vehículo a vehículo en redes ad hoc para entornos de red vehicular de alta velocidad, que establece que múltiples intercambios de datos/paquetes deben ser completados en un periodo de tiempo de 50 milisegundos (Khairnar & Kotecha, 2013).

El protocolo IEEE 802.11p también conocido como DSRC está diseñado para redes ad hoc. Actualmente este es el único estándar para comunicaciones directas V2V. Cabe destacar que, los estándares originales de DSRC fueron realizados en Europa, Japón y Corea (Khairnar & Kotecha, 2013).

Finalmente, V2V utiliza IEEE 802.11p como canal principal de comunicación, el cual surge como una corrección del estándar IEEE 802.11 para agregar Acceso de Red Inalámbrica a Ambientes Vehiculares (WAVE). Estas correcciones implican pequeñas modificaciones a la capa física y MAC del protocolo con el fin de alcanzar una conexión robusta y de respuesta rápida para vehículos en movimiento. (Bergenhem, 2012).

FIGURA 4.14: Pila de protocolos de DSRC. Khairnar and Pradhan (2013)



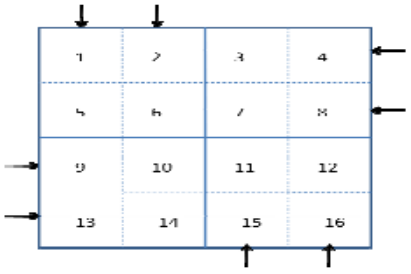
4.4. Manejo de intersecciones utilizando redes vehiculares

Manejo de intersecciones usando redes vehiculares es el trabajo realizado por Reza Azimi, Gaurav Bhatia, Ragunathan (Raj) Rajkumar y Priyantha Mudalige de la Universidad de Carnegie Mellon y la compañía automotriz General Motors. Este trabajo presenta dos protocolos de V2V destinados para intersecciones: Protocolo de Intersección de Máximo Progreso (MP-IP) y el Protocolo de Intersección de Cruce Concurrente (CC-IP). En el presente capítulo se describirá a detalle el protocolo MP-IP, el cual es el foco de estudio del presente trabajo.

Según Azimi *et al.* cerca del 23 % de colisiones alrededor del mundo ocurren en intersecciones, esto es alrededor de un millón de colisiones al año. El objetivo de su investigación es de aumentar la seguridad y rendimiento del tráfico en intersección utilizando cooperación asistida entre vehículos autónomos. Esto significa, que los vehículos autónomos puedan cooperar entre sí para la mejora del tráfico en intersecciones.

4.4.1. Detección de colisiones Se define intersección como un cuadrado perfecto que tiene puntos de entrada y salida para carril conectado a él, el área de intersección se considera como una cuadrícula dividida en celdas pequeñas. Cada celda en la intersección está asociada a un identificador único. La siguiente figura muestra una intersección con dos carriles y cuatro direcciones.

FIGURA 4.15: Rejilla de intersección dividida en pequeñas celdas

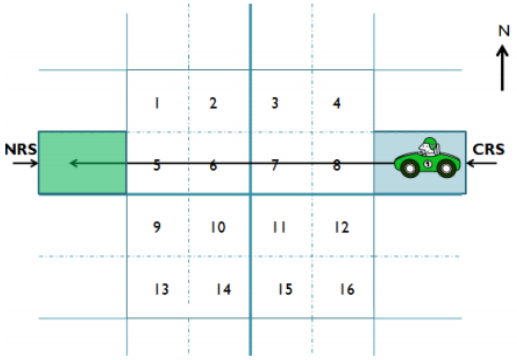


Se asume que cada vehículo tiene acceso a un mapa digital que provee información de la carretera y los carriles. Además, cada intersección está asociada a un identificador único en este mapa y posee carriles de salida y también cada vehículo tiene acceso a su posición global (GPS).

En el contexto de una intersección se define como segmento de vía actual (CRS) al segmento de carretera en donde se encuentra el vehículo previo a la intersección y segmento de vía siguiente (NRS) al segmento de carretera en donde el vehículo estará después de cruzar la intersección. Adicionalmente, se define la Tabla de Trayectorias de

Celdas (TCT) que determinan las celdas que serán ocupadas por el vehículo mientras cruza el área de intersección. TCT utiliza NRS, CRS y el número de carril como entradas para devolver una lista de celdas que a partir de ahora serán referidas como Lista de Trayectorias de Celdas (TCL). En el caso de la figura siguiente el TCL devolvería [8,7,6,5].

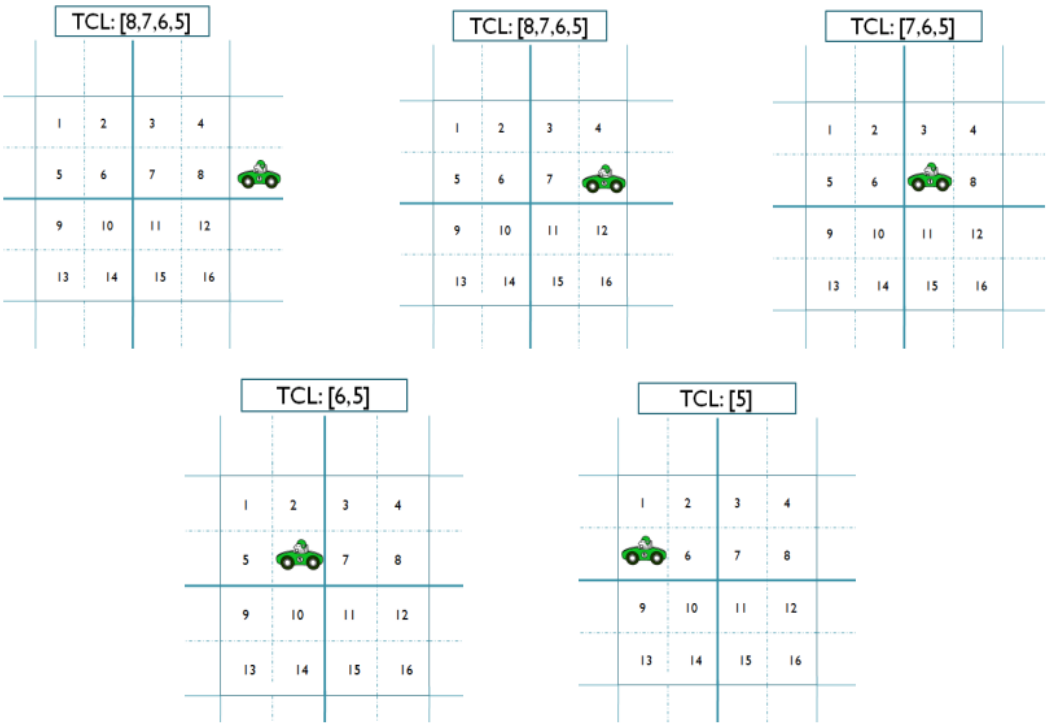
FIGURA 4.16: Ilustración de TLC, CRS y NRS



Es necesario para actualizar la TCL de forma exacta, que cada vehículo tenga conocimiento de la celda exacta que esté ocupando en dicho momento. Esta celda será utilizada para actualizar la TCL y luego será enviada a los vehículos circundantes como parte del mensaje de seguridad básica (BSM).

Para actualizar la TCL, cada vehículo determina el número de celda en el que esté ubicado utilizando GPS para mapear la celda. Si el vehículo detecta que no ha ingresado al área de intersección, no se modifica la TCL. En este caso, la TCL contiene una lista de números de celdas que corresponden a las celdas que ocupará el vehículo mientras cruza el área de intersección. Si el vehículo se encuentra en el área de intersección entonces utiliza la celda actual y modifica la TCL de la siguiente manera: dado que la TCL es una lista ordenada (con respecto a los números de celda) el vehículo puede informar en qué celdas ha circulado y en qué celdas está próximo a circular. Por lo tanto, el vehículo actualiza la TCL eliminando los números de celda que corresponden a las celdas que ya pasó completamente y la nueva TCL contiene el número de celda actual(en el que se encuentra ubicado el vehículo) con los números de celda restantes de la trayectoria. La siguiente figura muestra un ejemplo de un vehículo cruzando un área de intersección, actualizando la TCL mientras esté adentro y fuera de ella.

FIGURA 4.17: Secuencia de actualización de TCL

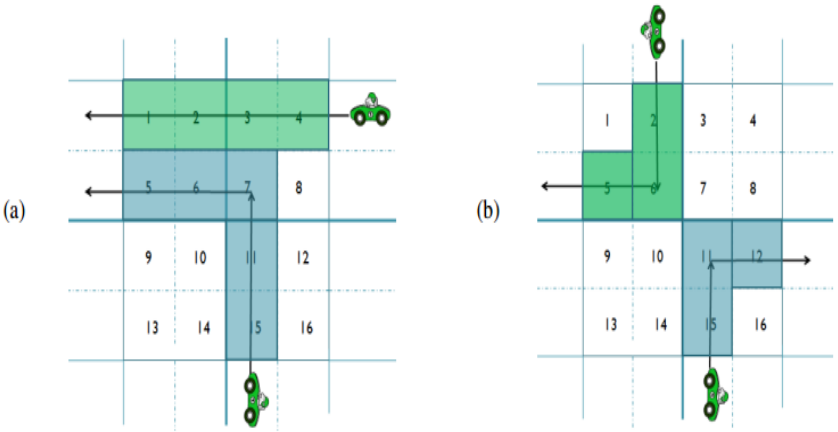


Una colisión ocurre en el área de intersección si dos o más vehículos tienen conflictos de tiempo y espacio. En otras palabras los vehículos tienen una colisión si sus intervalos (hora de entrada, hora de salida) se solapan y ocupan al menos una celda en común en sus trayectorias a lo largo de la intersección. Si cualquiera de estas dos condiciones es falsa, entonces no habrá conflictos y los vehículos podrán continuar con sus trayectorias de forma segura.

El algoritmo de detección de colisiones para intersecciones (CDAI) se ejecuta en todos los vehículos, este algoritmo hace uso de la información obtenida de los mensajes de seguridad entrantes que son difundidos por los vehículos que se encuentran a los alrededores. El algoritmo utiliza las TCL del vehículo emisor y receptor de los mensajes de seguridad, comparando ambas listas, para determinar si existe una celda en común a lo largo de las trayectorias mientras cruzan la intersección. Si el algoritmo detecta una potencial colisión, entonces retorna el primer número de celda en conflicto al que se llama Celda que intersecta la trayectoria (TIC).

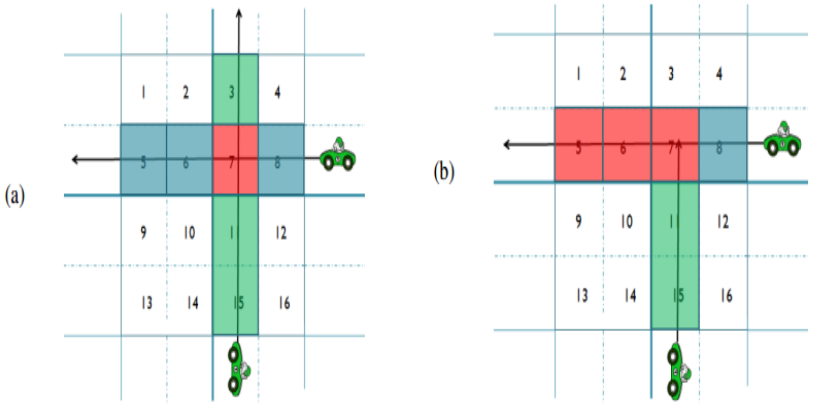
A continuación se muestran dos escenarios en los que dos vehículos están dentro del área de intersección pero ninguno tiene una celda en conflicto, TIC, ya que no ocupan celdas en común durante su trayectoria.

FIGURA 4.18: Escenarios en los que no existe conflictos de espacio entre dos vehículos



En la siguiente figura podemos observar un ejemplo en donde existe conflicto, ya que dos vehículos intentando cruzar el área de intersección comparten al menos una celda en su trayectoria. En este caso el algoritmo CDAI retorna la primera TIC, que en este caso es la celda número 7.

FIGURA 4.19: Escenarios ejemplo con conflictos de espacio



En este caso una colisión potencial es detectada por el algoritmo CDAI. En este caso se hace un algoritmo llamado: Primero en Entrar-Primero en ser Servido (FCFS), que se utiliza para asignar prioridades a los vehículos según van ingresando, un vehículo que ingresa antes que los demás puede cruzar el área de intersección antes que los demás. Si los vehículos ingresan al mismo tiempo al área de intersección, existe una política de prioridad que asigna mayores prioridades a los vehículos que ingresan por carretera (calle, avenida, vía, según sea el caso) primarias que a los vehículos que ingresan por carretera secundaria. Si aún así existe conflicto, el vehículo con mayor VIN (Número de

Identificación de Vehículo), tendrá más prioridad que el otro, el VIN es único para cada vehículo.

El CDAI se ejecuta en cada vehículo y basado en la información obtenida por medio de los mensajes V2V, decide si un vehículo puede cruzar el área de intersección de forma segura o puede avanzar hasta antes de una TIC para evitar colisiones.

4.4.2. Máximo Progreso-Protocolo de Intersección(MP-IP) MP-IP ha sido diseñado para aumentar el rendimiento de tráfico en intersecciones para que los vehículos puedan avanzar de forma segura sin ningún conflicto. Este nuevo protocolo V2V hace uso de las TCL y los algoritmos CDAI.

En este protocolo se definen tres tipos de mensajes:

- **ENTER** Es utilizado para informar a los vehículos vecinos que el vehículo se está acercando al área de intersección con intenciones específicas de cruzar. El mensaje de tipo ENTER se compone de los siguientes elementos: identificador de vehículo, Segmento de vía actual, carril actual, segmento de vía próximo, próximo vértice, hora de llegada, hora de salida, TCL, hora de llegada a las celdas, número de secuencia de mensaje y tipo de mensaje (en este caso ENTER).
- **CROSS**: Se utiliza para informar que el vehículo se encuentra dentro del área de intersección, este mensaje contiene los detalles de trayectoria del vehículo emisor, el cual identifica el espacio que ocupará el vehículo mientras cruce. El mensaje CROSS contiene los mismos elementos que el mensaje de tipo ENTER. Su TCL contiene una lista actualizada de las celdas de trayectoria, las horas de llegada relativas para la celda actual y las celdas restantes de la trayectoria en el área de intersección. También se adjunta el tipo que en este caso es CROSS.
- **EXIT**: El mensaje indica que el vehículo ha salido de los límites de intersección. El mensaje de tipo EXIT, contiene los siguientes parámetros: Identificador del vehículo, Número de secuencia de mensaje y Mensaje de tipo Exit.

Cada vehículo utiliza sus propias coordenadas GPS, velocidad y acceso al mapa en la base de datos para calcular la distancia a la intersección próxima y la distancia circulada de la intersección previa. Se consideran tres estados de intersección para cada vehículo basados en su posición relativa dentro del área de intersección.

- **Intersection-Approach**: Cuando la distancia del vehículo al área de intersección es menor que un valor umbral D_{ENTER}
- **Intersection-Enter**: Cuando un vehículo está dentro de los límites de la intersección.

- **Intersection-Exit**: Cuando el vehículo sale de la intersección, hasta que llega hasta a una distancia con un valor umbral D_{EXIT}

A continuación se describen los dos algoritmos para el envío y recepción de mensajes a través del protocolo MP-IP, para el vehículo A y B donde $A \neq B$.

Algoritmo 1: MP-IP, Vehículo receptor

Entrada: Mensaje de seguridad en intersección: RM

Salida : Movimiento del vehículo B en la intersección

```
1  begin
2      if (RM = ENTER or RM = CROSS) then
3          Ejecutar CDAI para detectar conflictos en la trayectoria con el vehículo A y
             encontrar TIC_AB
4          if TIC_AB = NULL then
5              Cruce la intersección
6          end
7          else
8              Ejecute política de prioridad FCFS
9              if Prioridad de B > Prioridad de A then
10                 Cruce la intersección
11             end
12             else
13                 Continue y pare antes de ingresar a TIC_AB
14             end
15         end
16     end
17     else if RM = EXIT then
18         if TIC_AB está vacía then
19             Cruce la intersección
20         end
21     end
22 end
```

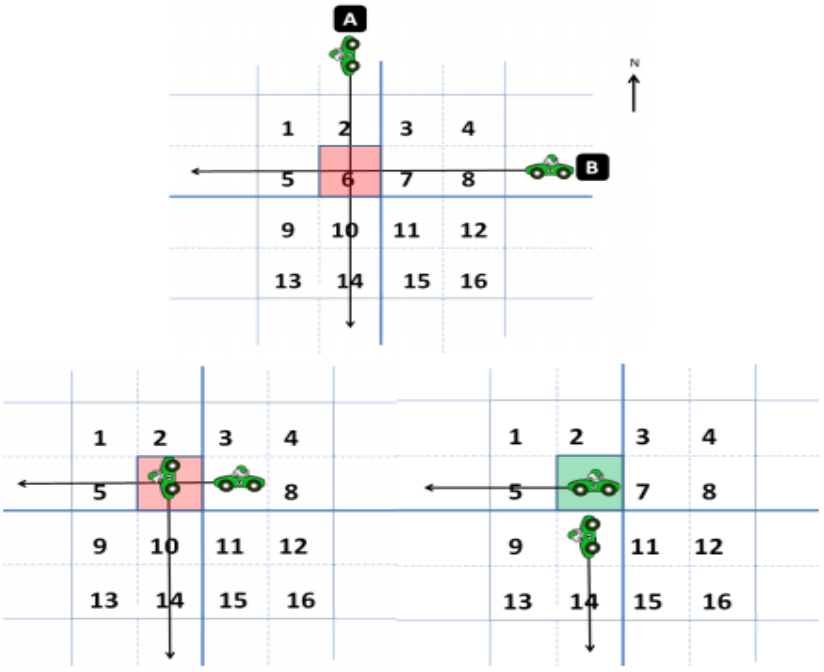
Algoritmo 2: MP-IP, Vehículo emisor

Entrada: Estado de intersección del vehículo
Salida : Difusión del mensaje de seguridad en intersección

```
1  begin
2      if ESTADO = Intersection-Approach then
3          Difundir mensaje tipo ENTER
4      end
5      else if ESTADO = Intersection-Enter then
6          Difundir mensaje tipo CROSS
7      end
8      else if ESTADO = Intersection-Exit then
9          Difundir mensaje tipo EXIT
10     end
11 end
```

Ahora mostramos el escenario de MP-IP. La siguiente figura muestra dos vehículos A y B acercándose a una intersección. Suponemos que el vehículo A tiene mayor prioridad que el vehículo B. En este caso el vehículo A tiene permiso de cruzar la intersección sin parar o disminuir su velocidad. El vehículo B continuará su trayectoria y parará antes de ingresar a la celda TIC con el vehículo A, la celda número 6. Cuando el vehículo A deja la celda 6, actualiza su propia TCL a [10,4] y envía un mensaje de tipo CROSS. Este informa al vehículo B que el TIC está libre y puede continuar su trayectoria pasando por la celda número 6.

FIGURA 4.20: Escenario de ejemplo de MP-IP



5. Metodología

5.1. Descripción

Azimir *et al.* propone en su trabajo *Protocolos de intersección utilizando redes vehiculares* el protocolo MP-IP. Los resultados de simulación del protocolo MP-IP muestran que su aplicación en escenarios de intersección en comparación al modelo de semáforos con 10 segundos de luz verde aumenta en 85.15 % el rendimiento del tráfico. A pesar de que los resultados de las simulaciones son favorables no existen pruebas que garanticen la total confiabilidad del protocolo para su implementación en un escenario real.

En este trabajo se presenta un análisis del protocolo MP-IP basado el método de verificación de modelos por medio de SPIN. La finalidad consiste en demostrar la correctitud del algoritmo en base a los propiedades que el protocolo debe cumplir; demostrar la correctitud del protocolo, nos garantiza confiabilidad del mismo para su posible implementación por terceras personas . Es importante notar que un error en el diseño del protocolo podría costar pérdidas humanas a la hora de su implementación, es por ello que vale la pena asegurarnos de la confiabilidad del protocolo.

A continuación se describe brevemente cada una de las fases para el estudio del protocolo MP-IP.

1. Se adaptó el modelo de comunicación Emisor-Receptor (Ver algoritmo 1 y 2 Sec. 1.4) del protocolo MP-IP al lenguaje de verificación de modelos PROMELA.
2. Se planteó un conjunto de propiedades que el protocolo debe satisfacer para garantizar su confiabilidad. Estas propiedades se plantearon formalmente utilizando Lógica Temporal Lineal, por esta razón, dichas propiedades se limitan a plantear proposiciones relativas a un tiempo futuro lineal.
3. Utilizando el modelo en PROMELA y las propiedades de LTL se verificó si el protocolo efectivamente satisface cada una de dichas propiedades por medio de SPIN. Además, se verificaron dos propiedades más: ausencia de bloqueo mutuo y código inalcanzable.

En las siguientes se hará hincapié en cada una de dichas fases.

5.2. Planteamiento del protocolo MP-IP en PROMELA

A partir en los algoritmos de Emisor-Receptor que propone Azimir et al. en su investigación se adaptaron al lenguaje de verificación de modelos PROMELA. Cada uno de estos algoritmos se planteó como un proceso independiente del otro, ambos se comunican por medio de paso de mensajes utilizando un único canal de comunicación de forma síncronica.

A continuación se muestra la implementación de los algoritmos Emisor y Receptor del protocolo MP-IP en PROMELA.

```
1  mtype = {enter, cross, exit};
2  mtype RM;
3  chan channel = [0] of {mtype};
4  bool TIC_IS_NULL;
5  bool priorityPolicy;
6  bool execPriorityPolicy
7  bool crossIntersectionME;
8  bool crossIntersectionOTHER;
9  byte state;
10
11 proctype sender()
12 {
13
14         if
15         :: state = 0 ->  channel!enter
16         :: state = 1 ->  channel!cross
17         :: state = 2 ->  TIC_IS_NULL = true; channel!exit
18         fi
19
20 }
21
22 proctype receiver()
23 {
24
25         channel?RM;
26
27         if
28         :: (RM == enter || RM == cross) ->
29                 if
30                 :: TIC_IS_NULL = true -> crossIntersectionME = true;
31                                          crossIntersectionOTHER = true;
32                 :: TIC_IS_NULL = false ->
33                         execPriorityPolicy = true;
34                         if
35                         :: priorityPolicy = true -> crossIntersectionME = true;
36                                                     crossIntersectionOTHER = false;
37                         :: priorityPolicy = false -> crossIntersectionME = false;
38                                                      crossIntersectionOTHER = true;
39                         fi
40                 fi
41         :: (RM == exit) ->
42                 if
43                 :: (TIC_IS_NULL == true) -> crossIntersectionME = true
44                 fi
45         fi
46
47 }
48
49 init{
50         atomic{
51                 run sender();
52                 run receiver();
53         }
54 }
```

A partir de acá suponga que existen dos vehículos A y B que se están comunicando entre sí. El proceso ***sender*** corresponde al vehículo A y el proceso ***receiver*** corresponde al vehículo B.

5.2.1. Descripción de variables A continuación se describe cada una de las variables utilizadas en el programa.

- **state** La variable state hace referencia al estado del vehículo A. El estado puede ser 0, 1 ó 2 que representan a los estados del vehículo en la intersección: Intersection-Approach, Intersection-Enter e Intersection-Exit respectivamente.
- **TIC_IS_NULL** Esta variable representa el valor booleano de la existencia de una celda en común entre las trayectorias de A y B durante la intersección. En otras palabras, si TIC_{AB} = NULL $\Rightarrow$ TIC_IS_NULL = true.
- **execPriorityPolicy** Variable booleana que representa la ejecución de la política de prioridad. Es verdadera en el caso de que la política de prioridad se ejecute para A y B.
- **priorityPolicy** Variable de tipo booleano. Su valor es verdadero si la prioridad del vehículo B es mayor a la del vehículo A.
- **crossIntersectionME** Esta variable es verdadera en el caso de que al vehículo B se le conceda permiso de avanzar por su trayectoria. Cuando es falsa implica que el vehículo B puede avanzar hasta antes de la celda TIC_{AB}
- **crossIntersectionOTHER** Esta variable es verdadera en el caso de que al vehículo A se le conceda permiso de avanzar por su trayectoria. Cuando es falsa implica que el vehículo A puede avanzar hasta antes de la celda TIC_{AB}
- **chanel** Representa la variable de tipo *chan* (canal) para la comunicación entre A y B.
- **enter, cross, exit** Constantes de tipo *mtype* representan los tipos de mensajes que puede enviar el vehículo A al vehículo B según su estado.
- **RM** Variable de tipo mtype cuyo valor puede ser: enter, cross y exit. Se utiliza para leer el mensaje de A a B.

5.2.2. Descripción de procesos El programa consiste de tres procesos: sender, receiver e init. A continuación se describe cada uno de ellos.

- **Sender** Representa el proceso del vehículo emisor de un mensaje. Por medio de *Lenguaje de Comandos Guardados* elige de forma no deterministica un estado para el vehículo A (línea 14-18). En función del estado emite el tipo de mensaje correspondiente al canal de comunicación. En el caso de que el estado del vehículo

sea 2 (Intersection-Exit) se limpia la celda TIC_{AB}, es decir se establece TIC_{AB} = NULL.

- **Receiver** Representa el proceso del vehículo receptor del mensaje. En la línea 25 realiza la lectura del mensaje y luego verifica el tipo de mensaje. Si el mensaje es de tipo enter o cross entonces es necesario chequear si TIC_{AB} = NULL, en el caso de que no lo sea se debe ejecutar la política de prioridad y en base a ella se concede paso únicamente a un vehículo. Si el mensaje es de tipo exit entonces se debe verificar si TIC_{AB} ya ha sido limpiada, si es nula entonces conceder el paso a B.
- **Init** Es el principal proceso del programa, es el encargado de instanciar los demás procesos. Ejecuta de forma atómica el proceso sender y receiver con el fin de ejecuar ambos procesos de forma secuencial, ya que es necesario que el proceso sender se realice antes que el proceso receiver para tener mensajes en el canal de comunicación.

5.3. Propiedades de Lógica Temporal Lineal

En esta sección se describe las propiedades que se plantearon para verificar el correcto funcionamiento del protocolo de comunicación vehicular. Estas propiedades están codificadas por medio de Lógica Temporal Lineal. Por lo que que el alcance de estas propiedades se limita a eventos que pueden ocurrir en un tiempo futuro lineal.

Las propiedades planteadas con su respectiva codificación en Lógica Temporal Lineal son las siguientes:

1. Si se ejecuta la política de prioridad entonces esta le concederá paso de manera seguida a solamente un vehículo y el otro vehículo avanzará hasta antes de TIC_{AB}

$$\begin{aligned} &\text{execPriorityPolicy} = \text{true} \Rightarrow \\ &\bigcirc((\text{crossIntersectionME} = \text{true } \textbf{and } \text{crossIntersectionOTHER} = \text{false}) \\ &\textbf{xor } (\text{crossIntersectionME} = \text{false } \textbf{and } \text{crossIntersectionOTHER} = \text{true})) \end{aligned} \tag{5.1}$$

2. Si la política de privacidad dictamina que la prioridad de A es menor a la de B entonces A no puede cruzar la calle inmediatamente.

$$(\text{priorityPolicy} = \text{false}) \Rightarrow \neg \bigcirc (\text{crossIntersectionOTHER} = \text{true}) \tag{5.2}$$

3. Siempre que dos vehículos A y B tengan una celda en común entre sus trayectorias entonces eventualmente se le concederá paso en la intersección a únicamente uno de los dos vehículos.

$$\begin{aligned} &\Box(\text{TIC\_IS\_NULL} = \text{false}) \Rightarrow \\ &\Diamond(\text{crossIntersectionME} = \text{true } \textbf{or } \text{crossIntersectionOTHER} = \text{true}) \end{aligned} \tag{5.3}$$

4. Cuando no existen celdas en conflicto entre dos vehículos A y B, se le debe conceder paso a ambos vehículos de forma seguida.

$$\begin{aligned} &(\text{TIC\_IS\_NULL} = \text{true}) \Rightarrow \\ &\bigcirc(\text{crossIntersectionME} = \text{true } \textbf{and } \text{crossIntersectionOTHER} = \text{true}) \end{aligned} \tag{5.4}$$

5. Si B recibe un mensaje de un vehículo que ha salido de la intersección, entonces B puede cruzar la intersección hasta que TIC_{AB} sea nula

$$\begin{aligned} &(\text{state} = \text{Intersection-Exit}) \Rightarrow \\ &(\text{crossIntersectionME} = \text{true } \mathbf{U} \text{ TIC\_IS\_NULL} = \text{true}) \end{aligned} \tag{5.5}$$

La codificación de estas propiedades en lenguaje PROMELA se muestra a continuación. Las propiedades se muestran en el mismo orden en las que describió.

```
1
2
3  ltl p1{
4
5
6          (execPriorityPolicy == true) -> X((crossIntersectionME == true &&
       ↪ crossIntersectionOTHER == false) ^ (crossIntersectionOTHER == true &&
       ↪ crossIntersectionME == false))
7
8  }
9  ltl p2{
10
11         (priorityPolicy == false) -> !X(crossIntersectionOTHER == true)
12
13 }
14 ltl p3{
15
16
17         [](( TIC_IS_NULL == false) ->  <>(crossIntersectionME == true  ^
       ↪ crossIntersectionOTHER == true))
18
19 }
20 ltl p4{
21
22
23         (TIC_IS_NULL == true) ->  X(crossIntersectionME == true  &&
       ↪ crossIntersectionOTHER == true)
24
25 }
26 ltl p5{
27
28         (state == 2) -> (crossIntersectionME == true U TIC_IS_NULL == true)
29
30 }
```

En el código anterior el símbolo X hace referencia al operador **Siguiente**. El símbolo [] hace referencia al operador **Siempre**. El símbolo hace referencia al operador **Eventualmente** y el símbolo U hace referencia al operador **Hasta**.

Adicionalmente se verificó la ausencia de código inalcanzable y de bloqueo mutuo en el modelo. Estas propiedades serán descritas en el capítulo de resultados.

6. Resultados

6.1. Búsqueda completa en el espacio de estados

SPIN provee varias opciones para la verificación de modelos. En este caso se utilizó SPIN para generar un verificador en código C, el cual realiza una búsqueda exhaustiva en todo el espacio de estados generado a partir del modelo en PROMELA para la verificación de la correctitud en las propiedades. En este caso se realizó una verificación de código inalcanzable y ausencia de bloqueo muturo en el modelo. Asimismo, se presentan otros elementos importantes como el total de estados en el espacio de estados completo, transiciones del mismo, memoria utilizada, entre otros. El comando de ejecución fue el siguiente:

spin -a modelo.pml

El cual generó código en C en un archivo llamado *pan.c*, el cual fue compilado utilizando gcc para generar un archivo objeto, cuya ejecución nos dejó el siguiente resultado:

FIGURA 6.1: Resultados de SPIN en búsqueda de espacio de estados

```
(Spin Version 6.4.0 -- 19 September 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 44 byte, depth reached 14, errors: 0
       49 states, stored
        2 states, matched
       51 transitions (= stored+matched)
        1 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.003       equivalent memory usage for states (stored*(State-vector + overh
ead))
    0.288       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage


unreached in proctype sender
        (0 of 10 states)
unreached in proctype receiver
        (0 of 25 states)
unreached in init
        (0 of 4 states)
```

A continuación se describe el significado de cada línea generado por el verificador.

(Spin Version 6.4.0 – 19 September 2014)

Identifica la versión de SPIN que generó el código pan.c que se utilizó para compilar el verificador.

+ Partial Order Reduction

El signo (+) indica que el algoritmo de *Reducción parcial de orden* fue utilizado en la búsqueda.

Full statespace search for:

Indica el tipo de búsqueda. La búsqueda por defecto es la búsqueda completa en todo el espacio de estados.

never-claim - (none specified)

El signo negativo (-) indica que ninguna propiedad de *never claim* o fórmula de LTL fue utilizada en esta ejecución.

assertion violations +

El signo (+) indica que la ejecución buscó violación de (*assertions*) especificadas por el usuario, en este caso no hay.

acceptance cycles - (not selected)

El signo negativo (-) indica que la ejecución no verificó la presencia de *non progress cycles.*

invalid endstates +

El signo (+) indica que se realizó una verificación de estados finales inválidos en el modelo (i.e ausencia de bloqueo mutuo).

State-vector 44 byte, depth reached 14, errors: 0

Cada estado requirió 32 bytes de memoria. La búsqueda en profundidad mayor contenía 14 transiciones desde la raíz del árbol (i.e el estado inicial del sistema). Ningún error fue encontrado en esta búsqueda.

49 states, stored

Un total de 49 estados fueron almacenados en el espacio de estados entero, cada uno representado por un vector de 32 bytes de memoria.

2 states, matched

En dos casos la búsqueda retornó a un estado ya visitado en el espacio de estados.

51 transitions (= stored+matched)

Un total de 51 transiciones fueron explorados en la búsqueda total del modelo.

1 atomic steps

Una de las transiciones fueron parte de una secuencia atómica.

128.730 total actual memory usage

Un total de 128.73 Megabytes fueron utilizados, incluyendo la pila, las tablas hash y todas las estructuras de datos relacionadas.

unreached in proctype sender (0 of 10 states)
unreached in proctype receiver (0 of 25 states)
unreached in init (0 of 4 states)

Indica que en los tres procesos no existió ningún estado que no fue visitado. De lo contrario mostraría el número de línea de la sentencia que fue inalcanzable. Esto significa que no existe código inalcanzable en el modelo.

6.2. Verificación de propiedades de LTL

En esta sección se describe los resultados de SPIN de las propiedades de LTL descritas en la sección 2.3. Las propiedades de LTL en PROMELA fueron agregadas al modelo inicial y nuevamente se eligió la opción $-a$ de SPIN para generar un verificador en lenguaje C, *pan.c*, el cual fue compilado nuevamente con gcc para generar un archivo objeto.

El archivo objeto fue ejecutado para cada propiedad con la opción -N que determinaba el nombre de la propiedad a analizar.

./objeto -N p1

Los resultados de cada ejecución se muestran en la siguiente tabla: (Ver detalles de cada ejecución en anexos 1-5)

Propiedad	**No. de errores**	**¿Se satisface?**
p1	0	Sí
p2	0	Sí
p3	0	Sí
p4	0	Sí
p5	0	Sí

7. Análisis de resultados

Los resultados muestran que no existen situaciones de código inalcanzable y de bloqueo mutuo en el protocolo. Por lo anterior podemos deducir que no existe alguna situación en la cual todos los procesos, en este caso vehículos, se queden bloqueados por la espera de alguna acción de otro proceso. Esto significa que un vehículo no se quedará en un mismo estado por tiempo infinito. Asímismo, el verificador no reportó nodos inalcanzables en ningún proceso del modelo en PROMELA, por lo tanto cada instrucción de ambos algoritmos puede ser potencialmente ejecutado a partir del estado inicial.

En cuanto a las propiedades de Lógica Temporal Lineal, no se reportaron errores en la ejecución de cada una de las propiedades, entonces se deduce que el modelo en PROMELA cumple con las propiedades planteadas en LTL.

Bajo el modelo en PROMELA y el conjunto de propiedades en LTL del protocolo MP-IP se concluye que este protocolo es correcto. Es importante mencionar que la correctitud del protocolo se centra específicamente en el diseño del mismo, por lo tanto, es posible asegurar que el protocolo MP-IP es correcto en cuanto a la secuencia de eventos lógicos de los algoritmos que definen el protocolo.

La correctitud del protocolo podría variar en función de las propiedades en LTL planteadas, es decir, si las propiedades fueran modificadas posiblemente el protocolo no seguiría siendo correcto. No obstante, las propiedades de código inalcanzable y bloqueo muto no se verían afectadas a menos que el modelo en PROMELA sea modificado.

Acerca de la herramienta de verificación de modelos, SPIN generó un verificador en C que funcionó eficientemente y el espacio de memoria utilizado fue de 128 Megabytes, esto a pesar de que el modelo en PROMELA haya generado solamente 49 estados.

8. Conclusiones

- Se plantearon un total de siete propiedades que sirvieron para determinar la correctitud del protocolo MP-IP.
- El modelo formal en PROMELA del protocolo MP-IP contempla los algoritmos de receptor y emisor de mensajes.
- Se aplicó el método de Verificación de Modelos al protocolo MP-IP utilizando la herramienta SPIN junto al modelo y propiedades en PROMELA.
- El verificador generado por SPIN mostró que el modelo en PROMELA del protocolo MP-IP satisface cada una de las cinco propiedades de Lógica Temporal Lineal planteadas.
- El protocolo MP-IP es libre de estados de bloqueo mutuo y de código inalcanzable según la exploración de estados del verificador generado por SPIN.
- El protocolo de comunicación vehicular MP-IP es formalmente correcto bajo el modelo y propiedades planteadas.

9. Recomendaciones

- Plantear propiedades de verificación formal utilizando otros tipos de Lógica Temporal, tales como: Lógica temporal por intervalos, Árboles lógicos computacionales y Lógica temporal métrica.
- Extender el trabajo para ambientes que incluyan vehículos autónomos y vehículos conducidos por humanos; además, se pueden incluir variantes en los tamaños de los vehículos (motocicletas, camiones) y no restringir a vehículos de un solo tamaño.
- Utilizar otro software de verificación formal alternativo a SPIN, en función del tipo de Lógica temporal que se desee utilizar.
- Realizar estudios de verificación formal de software utilizando otras investigaciones existentes de comunicación vehiculasr. Dichas estudios se pueden emplear a otros escenarios vehiculares como rotondas y carreteras.

10 Bibliografía

Anderson, J. M., Nidhi, K., Stanley, K. D., Sorensen, P., Samaras, C., and Oluwatola, O. A. (2014). *Autonomous vehicle technology: A guide for policymakers*. Rand Corporation.

Arnold, D. (2000). The explosion of the ariane 5. Disponible en `http://www-users.math.umn.edu/~arnold/disasters/ariane.html`. Accedido el 10 de Octubre, 2016].

Azimi, R., Bhatia, G., Rajkumar, R., and Mudalige, P. (2012). Intersection management using vehicular networks. Technical report, SAE Technical Paper.

Baier, C., Katoen, J.-P., and Larsen, K. G. (2008). *Principles of model checking*. MIT press.

Ben-Ari, M. (2008). *Principles of the Spin model checker*. Springer Science & Business Media.

Bergenhem, C., Hedin, E., and Skarin, D. (2012). Vehicle-to-vehicle communication for a platooning system. *Procedia-Social and Behavioral Sciences*, 48:1222–1233.

Bouchrika, I. (2013). Introduction to distributed systems. [En línea; accedido 10 de Octubre, 2016].

Dommety, G. and Jain, R. (1998). Potential networking applications of global positioning systems (gps). *arXiv preprint cs/9809079*.

Dowson, M. (1997). The ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes*, 22(2):84.

Garis, A. G. (2010). Lógica temporal en verificación de modelos de software: origen y evolución hasta tiempos actuales. *Fundamentos en humanidades*, (21):151–161.

Gómez, M. C. (2012). Pasarela software upnp para red x-10 en entorno domótico.

Harding, J., Powell, G., Yoon, R., Fikentscher, J., Doyle, C., Sade, D., Lukuc, M., Simons, J., and Wang, J. (2014). Vehicle-to-vehicle communications: Readiness of v2v technology for application. Technical report.

Howard, B. (2014). V2v: What are vehicle-to-vehicle communications and how do they work? `https://www.extremetech.com/extreme/`

176093-v2v-what-are-vehicle-to-vehicle-communications-and-how-does-it-work/. [En línea; accedido 10 de Octubre, 2016].

Huerta, E., Mangiaterra, A., and Noguera, G. (2005). Gps: posicionamiento satelital. *Rosario: UNR Editora, Universidad Nacional de Rosario*.

Khairnar, V. D. and Kotecha, K. (2013). Performance of vehicle-to-vehicle communication using ieee 802.11 p in vehicular ad-hoc network environment. *arXiv preprint arXiv:1304.3357*.

Khairnar, V. D. and Pradhan, S. N. (2013). Simulation based evaluation of highway road scenario between dsrc/802.11 p mac protocol and stdma for vehicle-to-vehicle communication.

Lafuente, A. (2011). Introducción a los sistemas distribuidos. *Universidad del País Vasco*.

Leveson, N. G. and Turner, C. S. (1993). An investigation of the therac-25 accidents. *Computer*, 26(7):18–41.

Li, Y. J. (2010). An overview of the dsrc/wave technology. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 544–558. Springer.

Lin, H. (1985). Software for ballistic missile defense.

Løvengreen, H. H. (2012). Introduction to spin.

Silva, M. (2004). Sistemas distribuidos. *Universidad Tecnológica Nacional de Mendoza*.

Toyota (2014). Toyota to bring vehicle-infrastructure cooperative systems to new models in 2015. [En línea; accedido 10 de Octubre, 2016].

Vázquez Martínez, L. (2013). La aventura de la exploración de marte.

11. Anexos

FIGURA 11.1: Resultado de ejecución de la propiedad 1 de LTL en Spin

```
cristian@ThinkPad-Edge-E545:promela$ ./pan -N p1
warning: never claim + accept labels requires -a flag to fully verify
pan: ltl formula p1

(Spin Version 6.4.0 -- 19 September 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p1)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 36 byte, depth reached 0, errors: 0
        1 states, stored
        0 states, matched
        1 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.280       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

FIGURA 11.2: Resultado de ejecución de la propiedad 2 de LTL en Spin

```
cristian@ThinkPad-Edge-E545:promela$ ./pan -N p2
warning: never claim + accept labels requires -a flag to fully verify
pan: ltl formula p2

(Spin Version 6.4.0 -- 19 September 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p2)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 52 byte, depth reached 3, errors: 0
        2 states, stored
        0 states, matched
        2 transitions (= stored+matched)
        1 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.280       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

FIGURA 11.3: Resultado de ejecución de la propiedad 3 de LTL en Spin

```
cristian@ThinkPad-Edge-E545:promela$ ./pan -N p3
warning: never claim + accept labels requires -a flag to fully verify
pan: ltl formula p3

(Spin Version 6.4.0 -- 19 September 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p3)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 52 byte, depth reached 27, errors: 0
       88 states, stored
       31 states, matched
      119 transitions (= stored+matched)
        2 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.007       equivalent memory usage for states (stored*(State-vector + overhead))
    0.280       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

FIGURA 11.4: Resultado de ejecución de la propiedad 4 de LTL en Spin

```
cristian@ThinkPad-Edge-E545:promela$ ./pan -N p4
warning: never claim + accept labels requires -a flag to fully verify
pan: ltl formula p4

(Spin Version 6.4.0 -- 19 September 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p4)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 36 byte, depth reached 0, errors: 0
        1 states, stored
        0 states, matched
        1 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.280       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

FIGURA 11.5: Resultado de ejecución de la propiedad 5 de LTL en Spin

```
cristian@ThinkPad-Edge-E545:promela$ ./pan -N p1
warning: never claim + accept labels requires -a flag to fully verify
pan: ltl formula p1

(Spin Version 6.4.0 -- 19 September 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p1)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 36 byte, depth reached 0, errors: 0
        1 states, stored
        0 states, matched
        1 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.280       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```