UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Planificación de trayectorias basada en verificación de modelos

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Pablo Fernández García

DIRECTOR DE TESIS

Dr. Ismael Everardo Bárcenas Patiño

Ciudad Universitaria, Cd. Mx., 2022

Reconocimientos

A mi familia, mamá, papá y hermanas:
Dedico esta tesis a las personas que siempre me alentaron a culminar una etapa más de mi trayectoria académica y de mi formación como profesionista.

A la fundación "Alberto y Dolores Andrade" y a Maggie Iriarte:
Gracias por todo el apoyo brindado, porque gracias a ustedes todas las puertas a mis proyectos siempre estuvieron abiertas, sin ustedes nunca lo hubiera logrado.

A la UNAM:
Gracias por todas las experiencias, conocimiento y oportunidades, todas estas me abrieron los ojos para conocer cómo es mi país y el mundo.

A mis director y revisores de tesis:
Gracias por el tiempo brindado y el apoyo demostrado para que finalmente pudiera lograr esta meta.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Pablo Fernández García. Ciudad Universitaria, Cd. Mx., 2022

Resumen

El presente documento muestra una nueva perspectiva al aborda el problema de la Planificación de Trayectorias, que consiste en la generación de una trayectoria, lo que es una ruta de un punto A a un punto B con ciertas restricciones. En la actualidad existen diferentes maneras de solucionar este, regularmente lo que buscan es la ruta o trayectoria más corta entre un punto y otro punto, pero no son capaces de seguir rutas cumpliendo ciertas especificaciones, lo cual se cumple con el enfoque propuesto de la Verificación de Modelos.

La Verificación de Modelos es un método en el que teniendo una estructura matemática relacional, se verifica si una fórmula lógica satisface una estructura para ser un modelo de la fórmula. Gracias a esta verificación se tiene la capacidad de contemplar variables dentro de la estructura y casos secuenciales que pueden darse dentro de esta, por ello su uso principal ha sido identificar errores dentro de los sistemas. Esta capacidad de describir un sistema para verificar posibles casos, plantea nuevas especificaciones como tener obstáculos o lograr un orden de pasos que ocurren en el sistema, por ello emplear este procedimiento permite verificar el cumplimiento de especificaciones para una ruta.

En esta tesis se muestra el acercamiento de la Verificación de Modelos para obtener trayectorias, donde mediante premisas se obtiene una conclusión o ruta. Estas premisas son determinadas por el lenguaje lógico de las lógicas modales, en específico el de la lógica temporal. Las lógicas modales con su complejidad computacional y poder de expresividad son capaces de especificar diversas restricciones, como lo es la espacialidad y la temporalidad, con ello se obtienen rutas que cumplen especificaciones más flexibles o estrictas a las actualmente usadas, por lo que logran dar nuevas aplicaciones a la Planificación de Trayectorias. Además, en el presente documento se muestra la aplicación de la Verificación de Modelos, con el desarrollo de un prototipo funcional de la Planificación de Trayectorias en un navegador de propósito general, usando un mapa con puntos a través del API de Google Maps para generar la estructura relacional, que dada una fórmula en CTL y su traducción al verificador NuSMV, entrega una ruta con especificaciones dadas por un usuario, con lo que se expone una aplicación a una situación cotidiana para planear una visita a Ciudad Universitaria.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Antecedentes

La Planificación de Trayectorias (PT) consiste en: dado un punto inicial y un punto final en un determinado mapa, encontrar una trayectoria que satisfaga un conjunto de restricciones. Una de las restricciones más comunes es la evasión de obstáculos. La PT tiene un sinnúmero de aplicaciones, que van desde la automatización de la navegación de agentes autónomos, tales como vehículos terrestres o aéreos (drones), los sistemas de navegación de automóviles, hasta la optimización de rutas, por mencionar algunos. Diferentes aplicaciones hacen uso de distintos algoritmos y combinaciones de ellos, como es el caso de uso del algoritmo de Djiskra o el A*, dando como respuesta una trayectoria, donde en la gran mayoría de los casos se busca la entrega de la ruta más corta (2). Aún cuando las empresas tienden a ser reservadas sobre los algoritmos que utilizan, en algunos casos se vuelve de dominio público por el conocimiento del desarrollador y su relación con la investigación (3).

En la Figura 1.1 se muestra un ejemplo de una trayectoria de un punto A a un punto B, en este caso de Ciudad Universitaria a las Torres de Satélite, donde dentro de la aplicación se pueden agregar otros destinos, empleando diferentes sistemas de transporte, pero que muestran como dentro de la aplicación, en este ejemplo de Google Maps, siempre siguen un orden definido de destinos.

La Verificación de Modelos (VM) se define en su forma más elemental desde el campo de la lógica matemática: determinar si una estructura satisface a cierta expresión en un lenguaje lógico. El término fue presentado por Clarke y Emerson (4) en 1981, para verificar una especificación, y así satisfacer una verdad en un sistema o verificar una falla, se obtiene a partir de un modelo, como abstracción del sistema, el cual es una estructura de Kripke, así llamadas por Saul A. Kripke (5), la cual se definirá más adelante y normalmente diseñada como una estructura que se representa como un sistema de transiciones dado el comportamiento de un programa o sistema, y este sistema consiste de un grafo,

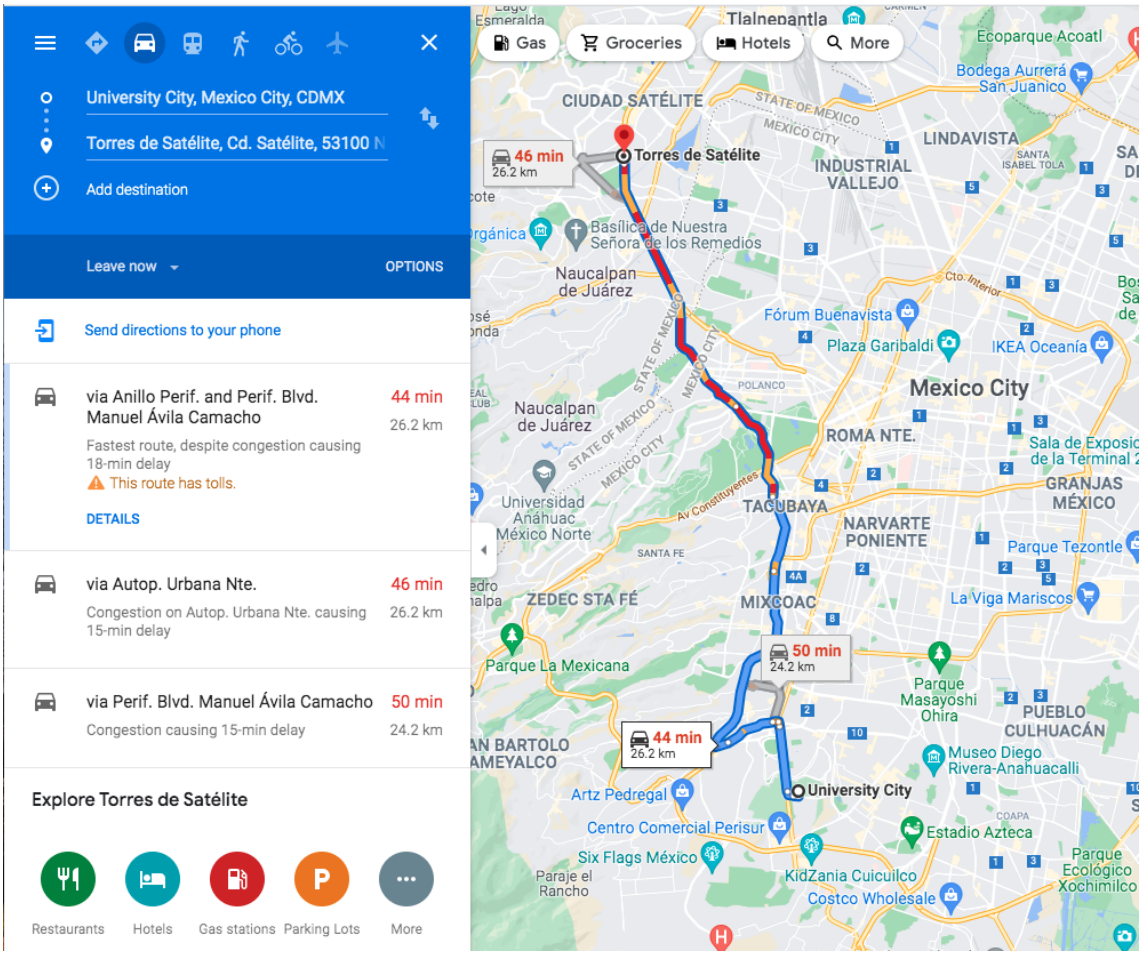


Figura 1.1. Planeación de rutas en Google Maps

cuyos nodos o vertices representan los estados alcanzables y aristas las cuales representan la transición a los estados, junto con una función de etiquetado que representa un conjunto de propiedades que se mantienen en ese estado. Un ejemplo de una estructura de Kripke se muestra en la Figura 1.2 con tres estados s_0, s_1, s_2 y proposiciones p,q y r.

En la actualidad existen diferentes métodos y algoritmos para generar trayectorias en un mapa, cada uno logra satisfacer diferentes necesidades, aunque el uso de herramientas de VM no es uno de los métodos que se utilice en la actualidad, permite dar flexibilidad al generar especificaciones, esto es gracias a la información contenida dentro de las estructuras de Kripke y a la secuencias en la lógica temporal, con esto se da respuesta para conseguir trayectorias o rutas, con una especificación adecuada.

La principal aplicación de la VM se ha hecho en la verificación formal de sistemas,

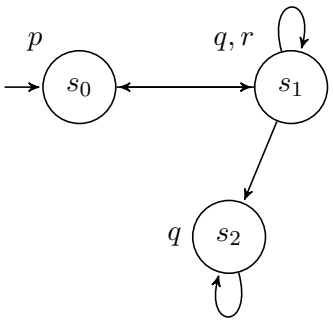


Figura 1.2. Ejemplo visual de una estructura de Kripke

software y hardware. En este dominio de aplicación en los sistemas, la estructura a usar es una representación abstracta de cierto sistema que se quiere analizar y la expresión lógica codifica la propiedad o comportamiento que deseamos verificar en el sistema. Una de las más grandes ventajas en la VM es la expresividad (6), término que se refiere a la amplitud de ideas que se pueden representar y comunicar en ese lenguaje. Cuanto más expresivo es un lenguaje, mayor es la variedad y cantidad de ideas que puede representar. Esta propiedad se utiliza en la lógica modal y en particular en la lógica temporal, al hacer uso de sus propiedades mostradas en sentencias que se representan en operadores y proposiciones en términos del tiempo. Ejemplo: "No voy a ser feliz *hasta* que viaje" o "*siempre* soy feliz al viajar". Aunque varios sistemas lógicos pueden enunciar estas sentencias, la lógica temporal es una manera sencilla de expresarlas, lo que muestra su poder de expresividad al señalar que una propiedad es implícita al modelo descrito, con lo que se interpreta que la propiedad es verdadera o falsa en cierto tiempo, y con esto que se cumpla en la correcta ejecución de dicho sistema o que no se logre cumplir en el sistema, esto es gracias al uso de operadores que son característicos dentro de esta lógica, con el uso de modalidades que se refieren a un momento en el tiempo, como es el caso del operador "hasta"' representado como U o "eventualmente"' representado como F, para verificar que efectivamente se alcanza un resultado.

Por ejemplo, dado un sistema imaginario con propiedades de estar ocupado *ocu*, con requerimiento *req* o en espera de un requerimiento, la especificación usando la fórmula F *ocu*, donde *ocu* es la propuesta a que el sistema eventualmente esté ocupado, este sistema es representado en la Figura 1.3, la especificación de esta propiedad se alcanza tanto en el estado s_1 como en el estado s_3 por lo que la especificación si se logra satisfacer dentro del sistema con una ruta.

Imaginando un sistema diferente, cuando una petición sea hecha, el acceso a un recurso eventualmente sea concedido, pero que nunca puede ser concedido simultaneamente con dos solicitantes. Estos casos muestran como alcanzar ciertos estados y buscar

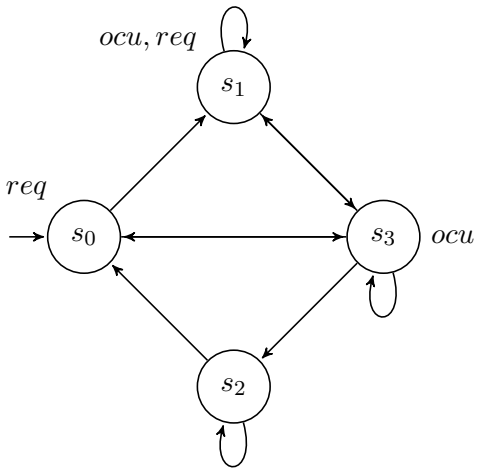


Figura 1.3. Representación de un sistema ejemplo de requerimientos en una estructura de Kripke

si se cumplen, al formular correctamente las proposiciones a cumplir, por lo que estas declaraciones muestran el poder de la expresividad de una fórmula con una proposición dentro de esta lógica y con el uso de herramientas de la VM, puede verificar si es posible llegar a este estado o no. Y con ello se expresan de una manera sencilla en un lenguaje.

Un ejemplo del poder de expresividad de la VM en la PT, se da a partir de un inicio y de una especificación, que es una fórmula a satisfacer, tal como es el caso de la demostración en el mapa de la Figura 1.1 por lo que con el uso de la lógica temporal y su operador eventualmente Fx siendo x= Torres de Satelite, como el lugar a eventualmente llegar, por lo que el llegar a este punto se muestra de manera simplista en el mapa, pero que cumple con la especificación para la PT.

Por ello en este trabajo se propone la aplicación de la VM para la PT. La estructura de Kripke es la representación de un mapa a través de una transformación y las restricciones que tiene que satisfacer para cumplir con una trayectoria, estas restricciones se codifican en términos de una expresión lógica. De manera específica se propone la construcción de un sistema de navegación a través de la VM en una estructura de Kripke. Con el uso de la VM se generan las trayectorias en un mapa y se logra dar flexibilidad a la respuesta para conseguir trayectorias o rutas con ciertas especificaciones, que se enuncian para obtener una fórmula, esta usarla para verificar si la estructura es un modelo de las especificaciones, tales como el eventualmente pasar por cierto lugar de interés, con ello conseguir rutas que satisfagan ciertas especificaciones a buscar y se logre obtener la ruta de los puntos por pasar, de inicio a final para cumplir con este propósito.

En este documento se muestran las bases para la generación de la estructura basada en un mapa y de la fórmula modal, con esto se genera una aplicación de la VM en un navegador de propósito general para obtener trayectorias, con la estructura y utilizando la lógica modal con la finalidad de dar especificaciones a la ruta, se tiene la capacidad de seleccionar puntos de inicio, pasos y obstáculos para obtener una estructura a utilizarse con la VM y con esto para conseguir rutas que cumplan las especificaciones deseadas, para verificar si existe una trayectoria satisfactoria y, si existe, mostrarla como resultado final, la cual se muestra gráficamente en un prototipo de interfaz de usuario con los lugares de interés dentro de un mapa y con el resultado.

1.2. Estructura de la tesis

El presente documento está formado por cinco capítulos con el siguiente orden:

El primer capítulo "Introducción" se presentan el problema, objetivos, estado del arte, alcance y contribuciones. En el segundo capítulo "Marco Teórico" se presenta el marco teórico donde se establecen y definen las bases de la VM, teoría de grafos, algunos tipos de lógicas modales, así como su semántica y sintaxis, ejemplos de cada una de ellas, y como se hace uso de esta lógica en la VM como herramienta para evaluar sistemas. El tercer capítulo "Planificación de trayectorias", se define el problema de la PT, se propone la construcción de la estructura y especificaciones con base en el marco teórico y como se hace el uso para la construcción de especificaciones dentro de la herramienta, además se muestra la implementación de la definición en en sistema de navegación de propósito general y el uso de la herramienta con NuSMV para la generación de especificaciones. En el cuarto capítulo "Conclusiones ", se muestran las conclusiones y principales usos que se podrían dar con la definición propuesta y el prototipo, así como futuras aplicaciones que se podrían producir siguiendo este trabajo.

1.3. Planteamiento del problema

En la actualidad para generar trayectorias en un navegador de propósito general, es decir una ruta de un punto inicial a un punto final, existen muchas soluciones dadas por diferentes algoritmos que cumplen esta labor tal como (7), cuyo enfoque general es el conseguir algoritmos eficientes para encontrar la ruta en el menor tiempo posible o en la menor distancia a recorrer, igual que en el trabajo (8), así al buscar encontrar la ruta más corta de diferentes maneras. Cada vez estas investigaciones se centran en lograr esta tarea más rapidamente tal como en (9), donde dependiendo del problema se usan diferentes algoritmos para su solución, así cada uno de estos trabajos usa un modelo que discretiza la representación de un mapa, de una manera fehaciente, para que facilite la evaluación y la implementación de los algoritmos para un resultado, pero que muchas veces no ofrecen una aplicación diferente a algo ya existente, ya que la mayoría de he-

rramientas no dan soluciones diferentes a la de la ruta más corta, lo que puede generar un nuevo uso en otras aplicaciones, por lo que el uso de la Verificación de Modelos logra darnos nuevas aplicaciones por la lógica que utiliza y que logra dar nuevos usos a la Planificación de Trayectorias.

El uso de la VM da flexibilidad al buscar cumplir con una ruta, por la expresividad de las lógicas usadas, tanto para ser estricta siendo más especifica o ser más relajada, algo que la diferencía a lo que existe en la actualidad, con ello se logra algo mayor a llegar de un punto a otro, como el seguir un orden o no seguirlo, tener obstáculos o no tenerlos, hacerlo adaptable a ciertas consideraciones y cumplir ciertas características, gracias a las diferentes operaciones con las que se cuenta, por basarse en las lógicas modales, esto trae consigo diferentes ventajas en comparación a los sistemas actuales como es la generación de restricciones más complejas y también al uso de especificaciones que pueden volverse más sencillas de expresar dentro del lenguaje de estas lógicas.

En este caso para aprovechar de la VM, es necesario un sistema para la construcción de la estructura de un mapa a usar, con la representación de ubicaciones o lugares dentro del sistema para la generación de rutas. Es necesario para crear una aplicación el conocer y programar una interfaz de usuario, que muestre de forma gráfica los lugares de interés de un mapa, conocer y hacer uso de lógica modal para la formulación de sus especificaciones, así con ello expresar las consideraciones de la trayectoria, para tener la capacidad de seleccionar puntos de inicio, paso u obstáculos para obtener una fórmula lógica, esta transformarla para que pueda utilizarse con la VM para obtener las rutas que cumplan las especificaciones deseadas y lograr conseguir una trayectoria satisfactoria.

1.4. Pregunta de investigación

¿Es posible generar trayectorias en un navegador de propósito general a través de la Verificación de Modelos?

1.4.1. Hipotesis

Es posible generar trayectorias en un determinado mapa a través de Verificación de Modelos, con la representación de un mapa en una estructura de Kripke y el uso la lógica temporal para realizar especificaciones de las trayectorias.

1.4.2. Objetivo general

Generar trayectorias en un mapa a través de Verificación de Modelos.

1.4.3. Objetivos específicos

- Desarrollar un algoritmo para la abstracción de un mapa de propósito general en términos de un modelo en un lenguaje lógico.
- Codificar rutas en un mapa de propósito general en términos de un lenguaje lógico.
- Desarrollar un prototipo con interfaz gráfica para generar trayectorias a partir de una herramienta de Verificación de Modelos.

1.5. Metodología

El objetivo general es necesario separarlo en los siguientes módulos, algoritmo para la construcción de la estructura, la codificación de rutas y el desarrollo del prototipo para validar gráficamente la trayectoria respuesta todo esto para facilitar su construcción. Por lo que primero se realiza la contrucción del modelo del mapa con sus ubicaciones a una estructura, haciendo uso de las estructuras de Kripke para la abstracción del mismo, usando la lógica temporal; se recomienda al lector si es de su interés (10) para el estudio de estructuras de Kripke, lógicas modales y la lógica temporal. Que es la forma en la que se entenderán las bases para generar la abstracción del mapa a una estructura y se podrá hacer uso de la VM (11), con ello se dará una función expresada en estas lógicas para especificar los lugares por pasar u obstáculos a evadir dentro de la técnica de la VM y la generación de la trayectoria con la misma, por último el desarrollo del prototipo para la aplicación real de la técnica para su visualización de forma gráfica.

Para lograr cada uno de los módulos en el problema es necesario realizar cada uno de los siguientes puntos:

- Estudio y conocimiento de la lógica modal para la generación del modelo.
- Diseño de la interfaz de usuario.
- Diseño de los algoritmos para la generación del modelo que representa al mapa, con base en una estructura de Kripke.
- Uso de la VM, especificamente del paquete NuSMV para obtener de rutas y especificaciones del modelo.
- Diseño del paquete NuSMV con la interfaz de usuario.
- Programación del sistema.
- Experimentación y respuestas.
- Conclusiones a la experimentación y respuestas.

Cada uno de estos puntos se mostrarán a mayor detalle en siguientes secciones del documento para la construcción de sistema estudiando de manera teórica la lógica modal, en particular la lógica temporal, para ser capaces de generar una estructura satisfactoria para la representación del mapa y también conocer así si es posible hacer uso de la VM para lograr nuestro objetivo, para ello se debe generar una fórmula que demuestre las especificaciones; en este documento se diseña un algoritmo para construir una estructura que logre representar mapas y pueda utilizarse para diferentes casos, de la misma manera se construye un algoritmo para facilitar el proceso de transformación para uso con la herramienta de NuSMV (12) en su implementación.

A su vez se construye y programa una interfaz con la API de Google Maps (13) para identificar visualmente el mapa a elegir, así como los lugares de interés a representar, generando a partir de estos puntos la fórmula para especificar la trayectoria a obtener, de esta manera también se añadió una forma de generar una especificación mediante una interfaz para añadir obstáculos y lugares por pasar, lo cual facilita el uso del sistema.

NuSMV no tiene una interfaz directa al sistema, por lo cual se creó una forma de comunicación para hacer uso de NuSMV y del sistema, simulando un servidor. Esto se programa y se añade como una forma para facilitar el uso de NuSMV y de obtener resultados sin tener que utilizar NuSMV de manera directa, con esto sé es capaz de generar diferentes modelos y especificaciones para probar el sistema, las respuestas y dar conclusión a nuestro objetivo inicial.

Con la finalidad de cumplir con un sistema integral que nos ayude a validar el objetivo inicial, se propone la construcción de un prototipo con la conexión de cada uno de los módulos presentados, para mostrar el resultado a nuestra hipotesis se muestra la experimentación de una estructura basada en Ciudad Universitaria, probando con NuSMV la Verificación de Modelos para validar y dar conclusión a la pregunta de investigación.

1.6. Motivación y estado del arte

En este apartado se realiza una revisión de aquellos trabajos que muestran similitudes y diferencias en el uso de VM para cumplir ciertos propósitos, formas de utilización de las herramientas de VM, además de otros trabajos parecidos en el uso de la VM a la tesis y que muestran las diferencias para lograr diferentes objetivos. Por lo que se muestran diferentes enfoques en el uso de la VM y herramientas, además de distintos problemas que se dan solución con el uso estas herramientas, para detallar el gran uso que tiene la VM.

Posteriormente se especifican las diversas técnicas y algoritmos que se encuentran en el verificador elegido (NuSMV). Y por ello con el objetivo de profundizar en el conocimiento de los aspectos relacionados con la VM y las lógicas modales, se han revisado

trabajos compatibles y el uso de algoritmos que cuentan estas herramientas para lograr ser un verificador de modelos.

Por último se expone el enfoque dado en la Planificación de Trayectorias, se añaden los trabajos y algoritmos relacionados a la Planificación de Trayectorias, que muestran el enfoque general que se le da a esa rama de la investigación, como buscan llegar a dar solución a problemas similares y que diferencía este trabajo de los demás.

1.6.1. Usos de la Verificación de Modelos

En la actualidad la gran mayoría de los trabajos que usan la VM buscan verificar si estructuras son modelos que pueden cumplir ciertas especificaciones, la gran mayoría ha sido usada en la verificación de sistemas de hardware y electrónicos, como en (14), donde se hace uso de un sistema para verificar circuitos síncronos, o en el uso de verificación de software y hardware como (15), la cual le ha dado multiples aplicaciones a la VM y diferentes herramientas que hacen uso de estas para proveer validación directa en los sistemas. También para casos de protocolos de redes como (16) donde de muestra la validación de seguridad y autenticación en el protocolo de control de transmisión TCP. Otros casos tienen que ver con modelos computacionales en sistemas distribuidos de control como en (17) y problemas de exclusion mutua.

También se han empezado a realizar diferentes usos, que no solo tienen que ver con lenguajes y hardware, sino con sistemas estocasticos y de probabilidad, haciendo uso de la "Lógica de computación de árbol" o por sus siglas en inglés CTL, que es una parte de la lógica temporal y nos ayuda a cuantificar, por sus operadores de uso que tan probable puede darse un evento, por ejemplo en cadenas de markov (11, p.745-780), esto lo logra con una nueva forma de CTL llamada PCTL que usa el lenguaje con cuantificadores y toma su probabilidad.

Con un enfoque diferente, también existen verificadores que buscan comprobar que un sistema en tiempo real, cuyo funcionamiento se hace con base en el tiempo, pueda cumplir con su función principal y no tenga fallas, sin comprometer un fallo que arruine un sistema que puede ser crítico, tal como (18), donde mediante de estos verificadores, se hace un modelo basada en lógica difusa, que controla un sistema de alumbramiento de las calles para evitar problemas de desperdicio y de accidentes, entonces con el verificador de modelos comprueba que no exista falla para su uso práctico.

En la VM existen con competencias anuales como es la *Hardware Model Checking Competition* por sus siglas HWMCC (19), donde diferentes universidades crean un verificador de modelos que logre cumplir con un problema de hardware y la herramienta más eficiente gana la competencia.

Cada uno de estos trabajos muestran el poder que tiene la VM para cumplir diferentes

funciones en distintos ámbitos.

1.6.2. El verificador NuSMV

Existen diferentes verificadores de modeles entre los cuales se encuentran: SPIN (20), NuSMV (1), CBMC y UPAAL. Cada verificador de modelos tiene sus ventajas y desventajas, diferencias estructurales de diseño y diferentes usos, esto debido tanto a su estructura interna como el objetivo por el cual fueron creados, donde su diferencia más clara es el tipo de lógica de la cual hace uso, así como su programación interna.

La herramienta propuesta a usar es NuSMV,que significan Nuevo Verificador de Modelos Simbólico, (*New Symbolic Model Verifier*) por sus siglas en inglés (12), como la herramienta de VM usada para el modelado y verificación, se detalla un poco acerca de esta; Se conoce que fue hecha alrededor del 2004 como reimplementación de SMV y nació de una unión de trabajo entre varias universidades, este sistema se basa en un proyecto OpenSource que deseaba que herramientas de uso de la VM fueran un proyecto más libre para el uso cientifico y menos en dominios de aplicación no tradicionales, o sea no industriales. Su nueva architectura se basa en una integración de Diagrama de Decision Booleana por las siglas en inglés OBDD y algoritmos para Problemas de Satisfabilidad Booleana que se conocen como SAT, como son el algoritmo Davis–Putnam–Logemann–Loveland, por las siglas de los nombres de los autores DPLL y que en matemática son problemas de complejidad NP-completos (21), así haciendo uso de estos algoritmos para resolver estos problemas para la verificación, se optimiza la respuesta a estas dificultades y con ello la herramienta es principalmente destinada a describir máquinas de estado finito.

En la Figura 1.4 se muestran los pasos para la construcción del modelo a verificar, lo que es la transformación realizada dentro de la herramienta con los pasos de la codificación del modelo, modelado, aplanamiento, simulación y manipulación de trazas, solucionadores a SAT para lograr la verificación de las especificaciones dadas en NuSMV y logrando la tarea de verificación dentro de la herramienta. En el caso del modelado del mapa deseado, esto sigue su forma más básica dentro de la herramienta, que es la de una máquina de estados finita o el de una estructura de Kripke, se verifica con el etiquetado de las propiedades, generada en una codificación booleana y con esto da respuesta a las especificaciones escritas dentro del lenguaje usado en la herramienta que pueden ser LTL, CTL o RTCTL.

1.6.3. Planificación de Trayectorias

La PT es un estudio que se ha dado en la actualidad desde muchos enfoques diferentes, ha sido muy estudiada desde la ciencia de la computación con el enfoque de encontrar algoritmos eficientes, hablando de complejidad computacional (21), esto es con la menor cantidad de recursos, tiempo y memoria, por ello van desde el lograr el menor tiempo

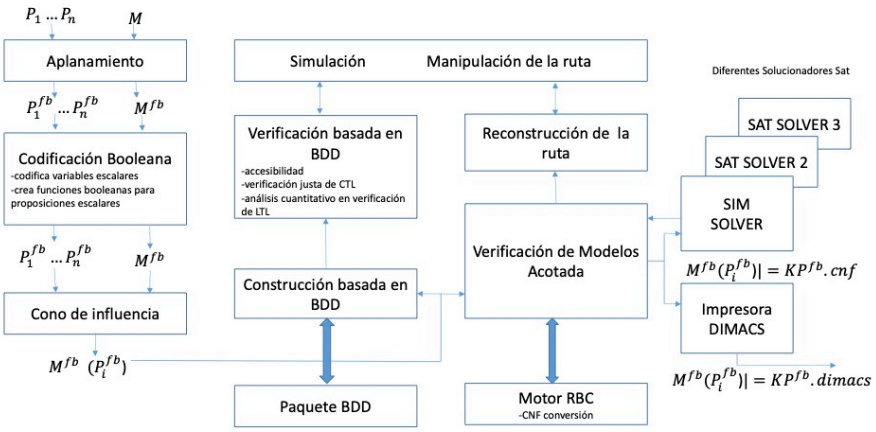


Figura 1.4. Estructura interna de NuSMV (1)

posible hasta el utilizar menos memoria. Estos algoritmos entregan diferentes respuestas dependiendo de su aplicación como lo son la entrega rápida de rutas en cuantos micro-segundos, incorporar información en tiempo real tomada de un servidor, entre otros.

Así para encontrar rutas en la actualidad, se tienen muchos avances de diferentes enfoques en esta área, que cuentan con muchas aplicaciones tratando de reducir la complejidad de los sistemas (2). Muchas veces estas soluciones son una combinación de algoritmos para tomar los beneficios de diferentes enfoques tales como la ruta más corta en tiempo o recorrido, el rendimiento de diferentes algoritmos en redes en tiempo real, el uso de diferentes sistemas de transporte o planeación basada en horario como la investigación de los algoritmos prácticos en estas rutas(9), etc.

Con el auge de la robótica y de los agentes autónomos, uno de los enfoques recientes es la generación de trayectorias para estos sistemas, por ello existen muchos estudios en como atacar el problema de moverse de un punto A a un punto B mientras se evade obstáculos y gracias a estos nuevos estudios, se han encontrado diferentes soluciones. En unos de los trabajos más recientemente estudiados se ha encontrado el uso de VM para el uso en navegación de agentes autónomos, como en (22) donde mediante el uso de la lógica temporal y su expresividad, logran modelar el controlador de viaje y trayectoria de un agente autónomo y que gracias a trabajos como este se tomó la inspiración para lidiar con un navegador de propósito general para la generación de trayectorias, con nuevos alcances a otro tipo de usuarios, por la expresividad lograda en la lógica usada, así por ser un tema sin muchas referencias en la actualidad, pero del cual se basa mucho del acercamiento a nuestro tema elegido, con el cual se decide aplicar en un navegador de propósito general, con una interfaz que hace uso en GoogleMaps para su visualización y con ello dar nuevas aplicaciones de uso.

1.7. Alcance y Limitaciones del trabajo

1.7.1. Alcance

- Realizar un generador de trayectorias dentro de un navegador de propósito general.

1.7.2. Limitaciones

- La complejidad de la estructura del mapa puede ser tan grande como el número de lugares de interés así como de variables que describan ese lugar, por lo que hacer la discretización del mapa puede ser extremadamente compleja en función de las etiquetas que deseen añadirse.
- La complejidad descriptiva para obtener una ruta puede crecer exponencialmente con el número de variables que se encuentren dentro del modelo.
- La complejidad de la especificación de trayectorias de acuerdo a la interfaz y al conocimiento del usuario, en especificaciones complejas.
- La funcionalidad limitada de desarrollo en el uso de una versión gratuita de Google Maps por el costo en su uso, tal como la obtención del modelo cada cierto tiempo.
- El uso de herramientas inexistente para lógicas todavía más expresivas como la es el cálculo-μ.
- Para la experimentación se esta limitado a la usabilidad de Google Maps.
- Dependencia a los cambios a futuro en Google Maps.

1.8. Contribuciones

Los objetivos son:

- Desarrollar un algoritmo para la abstracción de un mapa de propósito general a una estructura.
- Codificar rutas en un mapa de propósito general en términos de un lenguaje lógico.
- Desarrollar un prototipo con interfaz gráfica para generar trayectorias a partir de una herramienta de Verificación de Modelos

Para lograr estos objetivos, se define la PT en términos de VM, con ello se muestra una nueva manera de afrontar la PT y le añade un nuevo método para buscar rutas con especificaciones, por ello se diseña un algoritmo para facilitar la construcción de la estructura a usar, para su uso se muestra la forma de codificación de las especificaciones de la ruta de una manera amigable, por la forma de expresión, después en su traducción a CTL y luego se realiza otra traducción para su uso en el verificador NuSMV y así para la obtención de la respuesta, también se agrega una manera para adicionar una fórmula manual en CTL, si es que se conoce y desea una expresión con mayor complejidad, esto para no limitar la generación de una ruta más compleja.

Con la aplicación e implementación de la PT se muestra un prototipo de un navegador de propósito general basado en VM, el cual especificamente funciona eligiendo la localización de una ubicación dentro de un mapa, con puntos y lugares de interés, creado para la utilización del mapa en términos anteriormente descritos, y así con el desarrollo de una forma semiautomática de comunicación del sitio y del verificador, se obtiene la respuesta dada en la herramienta mostrando su visualización de gráfica en el mapa seleccionado, haciendo uso de la trayectoria obtenida, así como la interacción con el sistema para que su uso sea más amigable y visible al usuario, todo esto para lograr el objetivo general, ya que con el uso de la VM, se agregan funcionalidades que no se encuentran en herramientas actuales, lo que la vuelven una nueva forma para la aplicación de la PT siendo adaptable a la situación del uso del mapa, lo que logra ser una nueva forma de obtención de rutas.

Como en lo visto en los trabajos anteriormente presentados de la VM, actualmente no se le ha dado ningún uso como el presente en un navegador de propósito general sino en su uso para agentes autónomos, pero el que se presenta en este trabajo logra cubrir una amplia gama de nuevas funcionalidades a agregar a estos sistemas, esto por el diferente enfoque a su aplicación, por ello gracias a una de las grandes diferencias presentadas en la aplicación de la VM, con el uso del API (Interfaz de programación de aplicación) de Google Maps para la representación visual de resultados, así como de los puntos del mapa que son de nuestro interes, se logra tener una aplicación que podría ser una extensión aplicable a navegadores de propósito general tal como Google Maps, que actualmente no cuentan, por que haciendo uso de la VM en la PT, se amplia la flexibilidad o incrementa la rigurosidad es especificaciones para la obtención de una trayectoria que es capaz de tener obstáculos, seguir o no seguir un orden particular para pasar por los puntos, tener caminos cíclicos, entre otros. lo cual es un uso diferente al que actualmente cuentan la mayoría de los sistemas. Con esto se logran nuevos alcances y usos en un navegador, que nos pueden agregar nuevas aplicaciones a su uso, tales como planeadores de viaje para turismo, modelos de rutas de evacuación para emergencias, complemento a diversas aplicaciones ya existentes, modificación de rutas para uso de diferentes vehículos, PT con base en requerimientos, sistemas de planeación, entre otros y donde lo necesario para alcanzarlo es la creación de la estructura y la especificación de las trayectorias con base en una fórmula lógica.

Capítulo 2

Marco teórico

En este capítulo se exponen las bases matemáticas para sustentar el trabajo, por tal motivo se exponen definiciones y diferentes tipos de trabajos relacionadas con desarrollo de aplicación. Se tiene como base la teoría de grafos, diferentes tipos de lógicas, además de conceptos relacionados a la VM. También se presenta un tipo de logica modal, la logica temporal. Para cada lógica se define su sintaxis y su semántica, con ejemplos relacionados a cada una de ellas y la conexión que tienen con otras áreas de las matemáticas, con lo que se logra la conexión los temas aquí presentados que dan base a la Verificación de Modelos y al uso que se dará más adelante para su aplicación.

2.1. Teoría de grafos

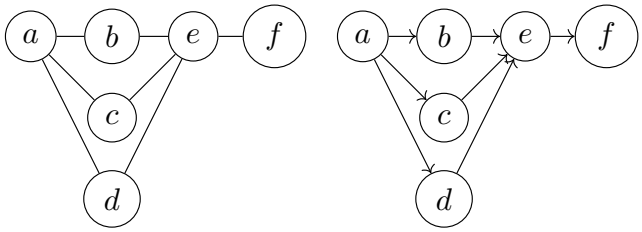
Los grafos son objetos matemáticos que representan una colección de un conjunto de puntos y líneas conectadas, o lo que es una representación gráfica de objetos y relaciones binarias entre estos (23).

Definición 2.1.1. (Grafo). Un grafo G esta definido como un par $G = (V, E)$, donde:

- V es un conjunto de vértices o nodos, y
- E es un conjunto de aristas o arcos, que relacionan estos nodos $E \subseteq [V]^2$

Con la notación $[V]^2$ denotamos el conjunto de todos los subconjuntos con dos elementos de V.

Ejemplo. Sea el grafo G con vértices $V = \{a, b, c, d, e, f\}$ y cuyas aristas $E = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, e\}, \{c, e\}, \{d, e\}, \{e, f\}\}$, su representación gráfica se puede observar en la Figura 2.2a.



(a) Grafo **(b)** Grafo dirigido

Figura 2.1. Ejemplos de grafos

Definición 2.1.2. (Grafo dirigido). Un grafo dirigido o digrafo es un grafo $D = (V, E)$ donde:

- $V \neq \emptyset$
- $E \subseteq (v_1, v_2) \in V^2$ es un conjunto de pares ordenados de elementos de V, donde v_1 es un vértice inicial ligado a un vértice final v_2, tal que $e \in E$, esta asociado a un par ordenado único de vértices v_1 y v_2 y se escribe $e = (v_1, v_2)$.
- Dada una arista (v_1, v_2), v_1 es su vértice inicial y v_2 su vértice final.

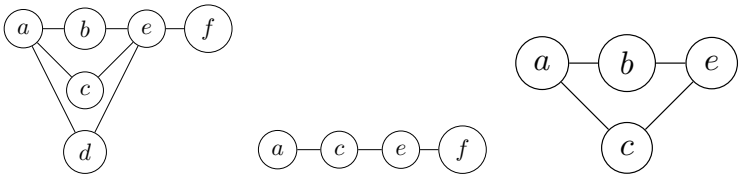
Para facilidad en ejemplos con una mayor cantidad de aristas E se define con la notación $(v_i : \{v_1, v_2, \ldots, v_j\})$ al conjunto de los pares ordenados tal que $(v_i : \{v_1, v_2, \ldots, v_j\}) = (v_i, v_1), (v_i, v_2), \ldots, (v_i, v_j)$.

Ejemplo. Sea el ejemplo el digrafo G con vértices $V = \{a, b, c, d, e, f\}$ y cuyas aristas $E = \{(a : \{b, d, e\}, (c, e), (d, e), (e, f)\}$, su representación gráfica se puede observar la Figura 2.1b

Definición 2.1.3. (Grado). Sea un grafo no vacío G, el número de vecinos de x en G denotado como $N(x)$ tal que $N(x) = \{y \in V | \{x, y\} \in E\}$ de modo que el grado del vértice x es el número de vecinos que tiene: $g(x) = |N(x)|$.

En un digrafo se considera dos tipos de grados, el interno y el externo, el interno expresa el número de adyacencias que recibe un nodo y el grado externo expresa el número de adyacencias que parten de un punto.

Definición 2.1.4. (Camino). Un camino o sendero en un grafo G es una secuencia de vértices $\rho = v_1, v_2, \ldots, v_k$ donde $(v_i, v_{i+k}) \in E$ tal que $\forall 1 \leq i \leq k$



(a) Grafo **(b)** Camino **(c)** Ciclo

Figura 2.2. Ejemplo de grafo con ciclo y camino

- $\rho = v_1, v_2, \ldots, v_{k-1}, v_k$ es un conjunto de elementos de V no necesariamente finito.

El número de vértices en el camino es su longitud y la longitud de ρ es denotada como ρ^k. Un camino es llamado simple sino repite vértices.

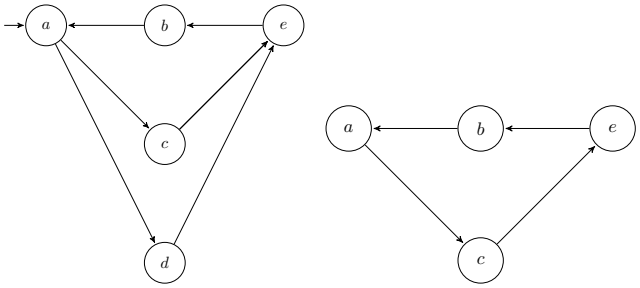
Ejemplo. Sea el grafo G con vértices $V = \{a, b, c, d, e, f\}$ y con aristas $E = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, e\}, \{c, e\}, \{d, e\}, \{e, f\}\}$, tiene un camino $\rho = a, c, e, f$ y ρ^4 cuyas representaciones se muestran en las Figuras 2.2.

Definición 2.1.5. (Grafo conexo). Un grafo conexo o conectado es un grafo en que todos sus vértices se encuentra en un camino.

Definición 2.1.6. (Ciclo). Un ciclo consiste en un camino cerrado, es decir, en el que no se repite ningún vértice, salvo el primero con el último. Un ciclo de n vértices se denota C_n, si $G = (V, E)$ es un ciclo C_n, el ciclo tiene n vértices $V = \{v_1, v_2, \ldots, v_n\}$ y n aristas formadas de la siguiente manera: $E = \{\{v_i, v_{i+1}\} | i = 1, ..., n-1\} \cup \{v_n, v_1\}$, por lo que, sea un grafo C y un camino $P = v_0, v_1, \ldots, v_k$ entonces el grafo $C = P + v_{k-1}v_k$ es llamado un ciclo. A la definición anterior también se le llama camino cerrado. Cualquier grafo con ciclos no es simple.

Ejemplo. Sea el grafo G, de la Definición 2.1.1, un ciclo C en G es $G = \{V = \{a, b, c, e\}, E = \{\{a, b\}, \{a, c\}, \{b, e\}, \{c, d\}\}\}$ cuya representación gráfica es la Figura 2.2c.

Ejemplo. Sea un ciclo dado el grafo dirigido G, de la Definición 2.1.2, un ciclo C en G es
$C = \{\{a, b, c, e\}, \{(a, b), (a, c), (a, d), (b, e), (c, e), (d, e)\}\}$ cuya representación gráfica se muestra en la Figura 2.3a.



(a) Grafo Dirigido **(b)** Ciclo en grafo

Figura 2.3. Ejemplos de grafo dirigido y ciclo

	v_1	v_2	v_3	v_j
v_1	$MA[1,1]$	$MA[1,2]$	$MA[1,3]$	$MA[1,j]$
v_2	$MA[2,2]$	$MA[2,2]$	$MA[2,3]$	$MA[2,j]$
v_3	$MA[3,3]$	$MA[3,2]$	$MA[3,3]$	$MA[3,j]$
v_i	$MA[i,1]$	$MA[i,2]$	$MA[i,3]$	$MA[i,j]$

Tabla 2.1. Matriz de adyacencia.

Un grafos se puede representar en una matriz de adyacencia o en una lista de adyacencia, dada la forma representar sus relaciones binarias.

Definición 2.1.7. (Matriz de adyacencia). MA: Sea una matriz de tamaño n^2 asociada a n cantidad de vértices, donde las filas y las columnas se identificar los vértices de grafo y cada casilla o celda representa $MA_{i,j} = 1$ si existe relación entre el vértice v_i y el vértice v_j del grafo y si no existe relación $MA_{i,j} = 0$.

Ejemplo. Sea que el ejemplo del grafo dirigido 2.1b la matriz de adyacencia es:

$$MA = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

En un grafo la matriz adyacencia obtenida es simétrica y en el digrafo no lo es.

Definición 2.1.8. (Lista de adyacencia). Lista de adyacencia (LA): Se utiliza un vector de tamaño n (elemento por cada vértice) donde $LA[i]$ almacena la referencia a una lista de los vértices adyacentes a v_i.

Ejemplo. La representación de la lista de adyacencia en el ejemplo del grafo dirigido 2.1b es:

a	b , c , d
b	e
c	e
d	e
e	f
f	

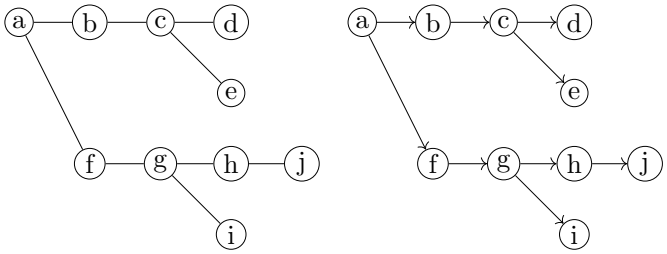
Definición 2.1.9. (Árbol). Un árbol es un grafo G que satisface cualquiera de estas condiciones alternativas:

- Cualquier par de vértices de G está conectado por exactamente un camino.
- G es conexo y no tiene ciclos.
- G no tiene ciclos y, si se añade alguna arista se forma un ciclo.
- G es conexo y si se le quita alguna arista deja de ser conexo.

Se identifica a un vértice como raíz o padre del cual deriva aristas a otros nodos que se llaman hijos de dicho padre o raíz. A los nodos que no tienen descendencia se les llama hojas.

Ejemplo. Sea el grafo $A = \{V, E\}$ donde:$V = \{a, b, c, e, f, g, h, i, j\}$ y $E = \{\{a, b\}, \{a, f\}, \{b, c\}, \{c, d\}, \{c, e\}, \{f, g\}, \{g, h\}, \{g, i\}, \{h, j\}\}\}$ la representación gráfica del árbol es la Figura 2.4a.

Ejemplo. Sea el grafo dirigido $A = \{V, E\}$ donde:$V = \{a, b, c, e, f, g, h, i, j\}$ y $E = \{(a : \{b, f\}), (b, c), (c : \{d, e\}), (f, g), (g : \{h, i\}), (h, j)\}$ la representación gráfica del árbol es la Figura 2.4b.



(a) Arbol **(b)** Arbol dirigido

Figura 2.4. Ejemplos de árboles

2.2. Lógica modal

En esta seccion se define la estructura de Kripke, la representación base en las lógicas modales, así como su sintaxis y semántica, además de ejemplos relacionados a cada una de ellas.

La lógica modal es el estudio formal de antiguas y nuevas modalidades, concebido desde la filosofía y luego fue presentado como un grafo con relación basada en semantica mostrada como lógica con fragmentos de primer y segundo orden (10).

Así desarrollada para representar fórmulas con proposiciones de necesidad y posibilidad. Por ello la fórmula $\Diamond P$ es leída como "posiblemente P" y $\Box P$ es leída como "necesariamente P". Por lo que la lógica modal puede ser usada para representar diferentes fenómenos dependendiendo de la consideración en la necesidad y la posibilidad.

En la lógica modal existen los operadores lógicos de la lógica de primer orden y con el trabajo de Kripke se consiguió la herramienta básica para el análisis semántico de esta lógica (5).

2.2.1. Estructuras de Kripke

Ahora, se abordan las estructuras de Kripke para la representación de modelos (10).

Definición 2.2.1. (Estructura de Kripke). Sea un conjunto de proposiciones atómicas AP, es decir expresiones booleanas sobre variables, constantes y predicados, que definen una estructura de Kripke sobre AP como una 4-tupla $K = \langle S, I, R, L \rangle$ donde:

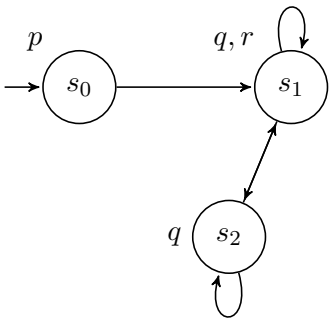


Figura 2.5. Ejemplo de una estructura de Kripke

- S es el conjunto de estados o nodos tal que no sea vacío.
- $I \subseteq S$, es el conjunto de estados iniciales.
- $R \subseteq S \times S$ es una relación de transición.
- L es una función de etiquetado (o interpretación) $L : S \to 2^{AP}$.

Ejemplo. Sea la representación gráfica de la estructura de Kripke la mostrada en la Figura 2.5, con tres estados s_0, s_1, s_2, sea $AP = \{p, q, r\}$ el conjunto de proposiciones atómicas y estructura de Kripke $K = \langle S, I, R, L \rangle$, donde:

- $S = \{s_0, s_1, s_2\}$.
- $I = \{s_0\}$.
- $R = \{(s_0, s_1), (s_1, s_1)(s_1, s_2), (s_2, s_1)(s_2, s_2)\}$.
- $L = \{(s_0, \{p\}), (s_1, \{q, r\}), (s_2, \{r\})\}$.

La función de etiquetado L define para cada estado $s \in S$ el conjunto $L(s)$ de todas las proposiciones atómicas que son válidas en s.

Al igual que en grafos dirigidos para facilidad en ejemplos con una mayor cantidad de relaciones en R se define con la notación $(s_i : \{s_1, s_2, \ldots, s_j\})$ al conjunto de los pares ordenados tal que $(s_i : \{s_1, s_2, \ldots, s_j\}) = (s_i, s_1), (s_i, s_2), \ldots, (s_i, s_j)$.

Definición 2.2.2. (Traza). Una traza de la estructura K es el conjunto de proposiciones de una secuencia de estados(camino) $\rho = s_1, s_2, \ldots, s_k$, tal que para cada $i > 0$, se cumple $R(s_i, s_{i+1})$. La traza sobre el camino ρ es la secuencia de conjuntos de proposiciones

atómicas $w = L(s_1), L(s_2), L(s_3)...$, que es una $\omega - traza$ sobre 2^{AP}.

Ejemplo. Sea la estructura K la mostrada en la Figura 2.5 puede producir el camino, $\rho = s_0, s_1, s_2, s_2, s_1, s_2$ y
$w = \{p\}, \{q, r\}, \{q, r\}, \{r\}, \{q, r\}$ es la traza de ejecución sobre dicho camino. K puede producir de AP el camino de $(\{p\}, (\{q, r\}, \{q\})^*)^\omega$ refiriendo como AP^ω al ciclo en el camino.

2.2.2. Sintaxis y semántica

Definición 2.2.3. (Sintaxis). El conjunto de fórmulas modales se define de la siguiente manera:

$$\phi := p \mid \top \mid \neg\phi \mid \phi \vee \phi \mid \Box\phi \mid \Diamond\phi$$

$$\text{donde } p \in AP.$$

Una fórmula modal es una combinación booleana de símbolos proposicionales que puede contener fórmulas dentro de ella de manera recursiva (10)

- p (Variable proposicional). $p \in AP$ y se utilizan para etiquetar los nodos.
- $\neg$ (Negación). $\neg\phi$ es la negación de ϕ. La negación de la fórmula se interpreta como el complemento de esta fórmula.
- $\top$ (Tautología). $\top$ se interpreta como la totalidad de los nodos.
- $\vee$ (Disyunción u o). $\vee$ es la interpretación de la unión de $\phi \vee \psi$.
- $\Diamond$(Fórmula modal existencial). $\Diamond\phi$ representa los nodos tales que: tienen al menos una adyacencia con un nodo denotado por ϕ.
- $\Box$ (Fórmula modal universal). $\Box\phi$ representa los nodos tales que tienen toda adyacencia con un nodo denotado por ϕ.

A continuación se describen la interpretación de las fórmulas respecto a una estructura de Kripke.

Definición 2.2.4. (Semántica). Dada una estructura $K = \langle S, I, R, L\rangle$ y un nodo $s \in S$, la semántica de las fórmulas se define de la siguiente forma:

- $[\![p]\!]_s^K = 1$ si y sólo si $p \in L(s)$

- $[\![\neg\phi]\!]_s^K = 1$ si y sólo si $[\![\phi]\!]_s^K \neq 1$
- $[\![\top]\!]_s^K = 1$
- $[\![\phi \vee \psi]\!]_s^K = 1$ si y sólo si $[\![\phi]\!]_s^K = 1$ o $[\![\psi]\!]_s^K = 1$
- $[\![\Box\psi]\!]_s^K = 1$ si y sólo si $\forall s' : (s, s') \in R$ implica $[\![\psi]\!]_{s'}^K = 1$
- $[\![\Diamond\psi]\!]_s^K = 1$ si y sólo si $\exists s' : (s, s') \in R$ y $[\![\psi]\!]_{s'}^K = 1$

Cabe destacar que con la sintaxis presentada es posible definir de diferentes formas la conjunción como:
$\phi \wedge \phi := \neg(\neg\phi \vee \neg\phi)$.

Ejemplo. Con lo definido en la semántica se presenta la fórmula:

$$\Box p \wedge \Diamond s \wedge \Diamond(r \wedge q)$$

En la estructura Kripke de la Figura 2.6 donde se observa que cada una de las restricciones al ser una conjunción se cumplen, tal que $\Box p$, en el nodo inicial se tiene p en todos los nodos adyacentes, después se exige que $\Diamond s$, lo cual cumple en el nodo s_2 y también que $\Diamond(r \wedge s)$ que dice que debe tener un nodo con r y q, que es s_1.

- $S = \{s_0, s_1, s_2, s_3\}$.
- $I = \{s_0\}$.
- $R = \{(s_0, s_1), (s_0, s_2), (s_0, s_3), (s_1, s_1), (s_2, s_2), (s_3, s_3)\}$.
- $L = \{(s_0, \{q\}), (s_1, \{p, q, r\}), (s_2, \{s, p\}), (s_3, \{p\})\}$.

Con ello K satisface un fórmula ϕ en s, cuando $[\![\phi]\!]_s^K = 1$, por lo que se define la satisfactibilidad de una fórmula ϕ en K donde la K es un modelo de ϕ, esto significa que después de analizarla bajo una interpretación la estructura K es verdadera, o lo que es lo mismo $[\![\phi]\!]_s^K = 1$ al ser así se dice que K es un modelo de ϕ, insatisfacible si ϕ no es satisfecha y válida (o tautología) si para cualquier interpretación del modelo la fórmula ϕ es verdadera.

2.2.3. Verificación de modelos

La tarea de la Verificación de Modelos se puede formular de manera local como (10):

Dada una estructura de Kripke K, un nodo s en K y una fórmula en lógica modal ϕ, si la estructura K puede satisfacer ϕ en s, entonces K es un modelo de ϕ:

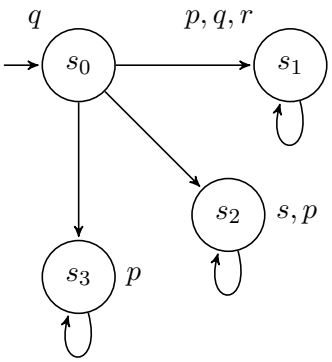


Figura 2.6. Ejemplo de una estructura de Kripke en la lógica modal

$$[\![\phi]\!]_s^K = 1$$

Se puede formular de manera global como:

Dada un estructura K y una fórmula en lógica modal ϕ, si la estructura K puede satisfacer ϕ en todos los nodos de S entonces K es un modelo de ϕ:

$$[\![\phi]\!]_S^K = 1$$

O se puede formular como una conjunción de ambas:

Dada un estructura de Kripke K y una fórmula ϕ, si existe un nodo $s \in S$, tal que $[\![\phi]\!]_s^K = 1$ se dice que K satisface ϕ, con eso también se dice que K es un modelo de ϕ. La verificación consiste en decidir si, dada una estructura K y una fórmula ϕ, K es o no un modelo de ϕ .

Por ello la Verificación de Modelos es en una tarea claramente computacional, donde una estructura es un almacén de información, que consiste de una colección de entidades con ciertas propiedades y relaciones entre ellas, una fórmula modal es una combinación construida recursivamente para verificar si estas propiedades se pueden satisfacer.

En este sentido usaremos la formulación local, así esto nos servirá tanto en la lógica modal como en cada una las siguientes lógicas que veremos para satisfacer las características de la planificación de trayectorias en un modelo dados ciertos estados iniciales.

Ejemplo. Dada la siguiente estructura K:

- $S = s_0, s_1, s_2, s_3$.
- $I = s_0$.
- $R = (s_0, s_1), (s_0, s_2), (s_0, s_3), (s_1, s_1), (s_2, s_2), (s_3, s_3)$.

- $L = (s_0, \{q\}), (s_1, \{p, q, r\}), (s_2, \{s, p\}), (s_3, \{p\})$.

Y la fórmula $\phi = \Box p \wedge \Diamond s \wedge \Diamond(r \wedge q)$

Entonces la VM buscará si existe la solución que verifique que K sea un modelo de la fórmula ϕ, la cual sólo existe en s_0, por ello $[\![\phi]\!]^K_{s_0} = 1$ y es un modelo en ese estado.

2.2.4. Contraejemplos

La prueba por contraejemplo es un concepto matematico antiguo, dada una cierta propiedad ϕ que se mantega para cada elemento en un conjunto S puede ser rechazado si existe un solo elemento $s \in S$ donde ϕ no sea verdadero (24).

Por lo que sea un problema satisfactibilidad en lógica, o sea el que resuelva un problema de decisión cuando decide si pertenece y no pertenece. Y sea como ejemplo A y B, donde estén relacionadas tal que $f = (A \vee B) \wedge \neg B$ donde $A = \top$ y $B = \bot$ al evaluarla se debe sustituir en la fórmula alguno de sus valores, por lo que siendo esta sustitución como $\bot$ el resultado es $\top$ y satisface el problema cuando es B es una contradicción.

Una de las características más importantes en la Verificación de Modelos es la habilidad para encontrar contraejemplos, dada esta habilidad el verificador determina si un modelo es verificable para una fórmula o es falsa, con esta encontrara el camino computacional que demuestre que la negación de esta fórmula es verdadera.

En el dominio de las aplicaciones de verificación de modelos, si una estructura resulta no ser un modelo de cierta fórmula, un contraejemplo representa una explicación del porqué la estructura no es un modelo. Por ello, si el conjunto de formulas ϕ es consistente o satisfacible, existe una interpretación, bajo la cual todas las fórmulas del conjunto son verdaderas, por lo que es un modelo pero en el caso contrario buscará un contraejemplo.

Definición 2.2.5. (Contraejemplo). Dado una estructura de Kripke K y un una fórmula modal ϕ, un contraejemplo C es un camino de $s, \{p_1, p_2, \ldots, p_m\}$, tal que $\{p_1, .., p_m\} = L(s)$ y $[\![\neg\phi]\!]^K_s = 1$ o alternativamente $L' \subset L$ tal que $(s, \{p_1, \ldots\}) \in L'$ y $[\![\neg\phi]\!]^K_s = 1$

Con ello se establece que la estructura K que no satisface ϕ.

Ejemplo. Sea la siguiente estructura K:

- $S = \{s_0, s_1, s_2, s_3\}$.
- $I = \{s_0\}$.
- $R = \{(s_0, s_1), (s_0, s_2), (s_0, s_3), (s_1, s_1), (s_2, s_2), (s_3, s_3)\}$.

- $L = \{(s_0, \{q\}), (s_1, \{q\}), (s_2, \{s, p\}), (s_3, \{p\})\}$.

Y la fórmula $\phi = \Box p$, un contraejemplo es cualquier camino donde no sea una solución para que sea un modelo, como el camino $\rho = s_0, (s_1)^\omega$, porque la propiedad p no se cumple indefinidamente por ello $[\![\Box p]\!]^K_{s_0} = 0$ por lo que el contraejemplo es $C = (s_0, \{q\}), (s_1, \{q\})^\omega$

2.3. Lógica temporal

En esta sección se presenta la lógica temporal, que extiende a la lógica proposicional clásica y es una lógica modal, con un conjunto de operadores temporales para la navegación entre relaciones. Se presenta también a CTL, así como su sintaxis y semántica, además de ejemplos relacionados a cada una de ellas. En la lógica temporal de igual manera existen los operadores lógicos usuales $\neg, \vee, \wedge, \Rightarrow$ y $\Leftrightarrow$. Una ruta o camino ρ en la estructura K es una secuencia no finita de estados $\rho = s_0, s_1, s_2, ..., s_i$ para una ruta.

2.3.1. TL

Lógica temporal y por sus las siglas en inglés de *Temporal Logic* se puede extender como una extensión a la lógica clásica, en ella emplean flujos de tiempo, relaciones y transiciones entre estados computacionales.

La lógica temporal (TL) es una línea construida por un conjunto AP es decir proposiciones atómicas y se rigen bajo conexiones booleanas, en una estructura K. Son utilizadas para definir sistemas donde no es necesario colocar el tiempo explícitamente. Utilizadas para describir secuencias de transiciones entre estados. Esta lógica tiene dos tipos: unario y binario. Los operadores básicos son el operador unario es X (siguiente), y un operador binario U (hasta), todos los demás se pueden representar a partir de estos dos operadores.

Haciendo uso de una estructura de Kripke K, ahora se presenta la sintaxis y semántica de la lógica temporal.

2.3.1.1. Sintaxis y Semántica

Definición 2.3.1. (Sintaxis). El conjunto de fórmulas de la lógica temporal se define de la siguiente manera, recursivamente:

$$\phi := p \mid \top \mid \bot \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid X\phi \mid \phi \text{ U } \phi.$$

Donde $p \in AP$.

Definición 2.3.2. (Semántica). Dada una estructura $K = \langle S, I, R, L \rangle$ y un nodo $s \in S$, la semántica de las fórmulas se define de la siguiente forma:

- $[\![p]\!]_s^K = 1$ si y sólo si $p \in L(s)$

- $[\![\neg\phi]\!]_s^K = 1$ si y sólo si $[\![\phi]\!]_s^K \neq 1$
- $[\![\top]\!]_s^K = 1$
- $[\![X\phi]\!]_s^K = 1$ si y sólo si $\exists s' : (s, s') \in R$ y $[\![\phi]\!]_{s'}^K = 1$
- $[\![\phi \text{ U } \psi]\!]_s^K = 1$ si y sólo si $\exists s_k : [\![\psi]\!]_{s_k}^K = 1$, $\forall s_i : (s_i, s_{i+1}) \in R$ y $[\![\phi]\!]_{s_i}^K = 1$ donde $0 \leq i \leq k$

La TL se construye a partir de un conjunto finito de variables proposicionales AP, los operadores lógicos $\neg$ y $\vee$, y los operadores modales temporales X (algunas publicaciones usan O o N) y U. Formalmente, el conjunto de fórmulas TL sobre AP se define inductivamente de la siguiente manera:

Si $p \in AP$ entonces p es una formula de TL; si ψ yϕ son formulas de TL entonces $\psi, \phi \vee \psi, X\psi$, y ϕ U ψ son formulas TL. X se lee como siguiente y U se lee como hasta. Aparte de estos operadores fundamentales, hay operadores lógicos y temporales adicionales definidos en términos de los operadores para escribir fórmulas TL de forma concisa. Los operadores lógicos adicionales son $\wedge$, $\Rightarrow$, $\Leftrightarrow$, verdadero $\top$ y falso $\bot$.

Además de los siguiente operadores temporales que se forman de los anteriores

- G para siempre (globalmente) $G\psi :=\bot$ R $\psi = \neg F \neg\psi$
- F para finalmente o eventualmente $F\psi := \top$ U ψ
- R para *release* ϕ R $\psi := \phi W(\phi \wedge \psi)$
- W para hasta debil ψ W $\phi := (\psi U \phi) \vee \mathrm{G}\psi = \psi U(\phi \vee \mathrm{G}\psi) = \phi R(\phi \vee \psi)$
- M para *release* fuerte. ψ M $\phi := \neg(\neg\psi W \neg\phi) = (\psi R\phi) \wedge F\psi = \psi R(\phi \wedge F\psi) = \phi U(\psi \wedge \phi)$

Se muestra la tabla 2.2 con los operadores principales de TL y la estructura de Kripke de forma gráfica de las mismas.

Ejemplo. Dada la Definición 2.3.1 se presenta una línea temporal donde p existe hasta que q exista en la línea del tiempo, siguiente s y que r esté eventualmente presente. Con estas condiciones se presenta la fórmula:

$$Fr \wedge q \text{ U } p \wedge Xs$$

Como ya se definió la satisfactibilidad, dada una estructura K puede satisfacer un fórmula ϕ en s con varias soluciones logrando así ser un modelo. Con esto podemos definir la satisfactibilidad en TL de la misma manera donde K satisface una fórmula ϕ

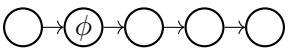
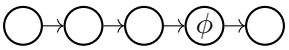
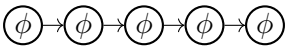
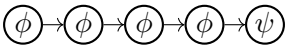
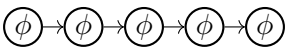
Operadores en lógica temporal	
Simbolismo	Explicación
$X\phi$	ϕ tiene que sostenerse en el siguiente estado. 
$F\phi$	ϕ tiene que sostenerse en el alguna parte del camino. 
$G\phi$	ϕ tiene que sostenerse en la parte siguiente del camino. 
ϕ U ψ	ψ tiene que sostenerse por lo menos hasta ϕ en el que se sostiene en una futura posición. 
ϕ R ψ	ϕ tiene que ser verdadero hasta e incluido el nodo donde ψ tiene que ser verdadero; si ψ no se vuelve verdadero ,ϕ siempre se sostiene.  

Tabla 2.2. Simbolismo en operadores lógica temporal

en s y por ello un modelo, esto significa que después de analizarla bajo una interpretación en nuestro modelo K es verdadera, o lo que es lo mismo $[\![\phi]\!]^K_{S'} = 1$ o $[\![s]\!]^K_{S'} = \phi$. Entonces para la estructura K que satisface la fórmula ϕ que fue ejemplo $Fr \wedge q$ U $p \wedge Xs$ será:

- $S = \{s_0, s_1, s_2, s_3\}$.
- $I = \{s_0\}$.
- $R = \{(s_0, s_1), (s_1, s_2)(s_2, s_3), (s_3, s_0)\}$.
- $L = \{(s_0, \{\ q\}), (s_1, \{p, q, r\}), (s_2, \{s, p\}), (s_3, \{p\})\}$.

Por lo que el camino $\rho = s_0, s_1, s_2, (s_3)^\omega$, su traza $\omega = q, \{p, q, s\}, \{s, p\}, \{p\}^\omega$ y un contraejemplo para mostrar que la fórmula $\phi = Gs$ no satisface, es el siguiente $C = (s_0, \{q\}), (s_1, \{p, q, s\}), (s_2, \{s, p\}), (s_3, \{p\})^\omega$.

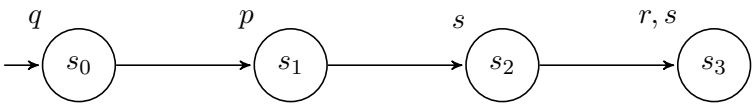


Figura 2.7. Ejemplo de gráfico de la fórmula TL: $Fr \wedge q$ U $p \wedge Xs$

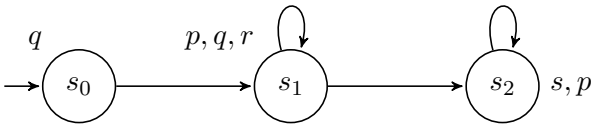


Figura 2.8. Estructura de Kripke para ejemplo en CTL

2.3.2. CTL

La lógica de árbol de computación, de las siglas en inglés de *Computational Tree Logic*, CTL utiliza modelos de árbol 2.1.9 con nodos finitos para representar caminos, por ello su nombre. Muestra una lógica de proposiciones de temporalidad lógica con una explicita cuantificación en futuros probables en un estado.

Estos cuantificadores son A para representar todos los caminos y para algún camino. Por lo tanto, cada fórmula con operadores temporales debe estar precedida por alguno de los cuantificadores A o E.

La representación en árbol de CTL con la Figura 2.8 da el árbol ejemplo 2.9,

- $S = \{s_0, s_1, s_2\}$.
- $I = \{s_0\}$.
- $R = \{(s_0, s_1), (s_1, s_1)(s_1, s_2), (s_2, s_2)\}$.
- $L = \{(s_0, \{q\}), (s_1, \{p, q, r\}), (s_2, \{s, p\})\}$.

Definición 2.3.3. Sintaxis. El conjunto de fórmulas de CTL se define de la siguiente manera recursivamente:

$$\phi := p \mid \top \mid \neg\phi \mid \phi \wedge \psi \mid \psi \vee \psi \mid EX\phi \mid AX\phi \mid E\phi \text{ U } \phi \mid A\phi \text{ U } \phi \mid E\psi \text{ R } \phi \mid A\psi \text{ R } \phi$$

donde $p \in AP$.

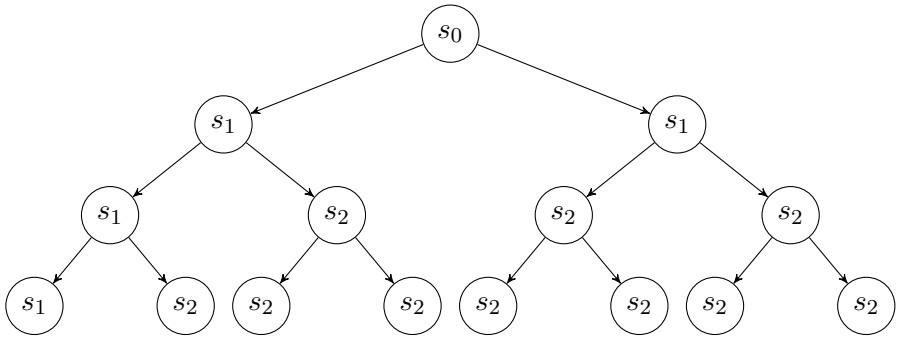


Figura 2.9. Representación gráfica del árbol CTL

- A (Operador de Inevitabilidad). Significa 'en todos los caminos' (inevitablemente)
- E (Operador Existencial). Significa 'existe por lo menos un camino' (posiblemente)

Junto con estos operadores en fórmula de CTL pueden hacer uso de las constantes booleanas de verdadero y falso.

Ahora presentamos la semántica de la lógica CTL y se describe su estructura K interpretadas sobre una estructura de árbol.

Definición 2.3.4. (Semántica). Dada una estructura de Kripke $K = \langle S, I, R, L\rangle$ donde R es una relación total, o sea para todo $s \in S$ existe un estado $s' \in S$ tal que $(s, s') \in R$, se dice que una CTL ϕ en K en un estado s y satisface $[\![p]\!]_s^K = 1$. Las fórmulas de CTL son interpretadas de la siguiente manera:

$[\![p]\!]_s^K = 1$ si y sólo si $p \in L(s)$

$[\![\neg\phi]\!]_s^K = 1$ si y sólo si $[\![\phi]\!]_s^K \neq 1$

$[\![\top]\!]_s^K = 1$

$[\![EX\phi]\!]_s^K = 1$ si y sólo si $\exists s' : (s, s') \in R$ y $[\![\phi]\!]_{s'}^K = 1$

$[\![AX\phi]\!]_s^K = 1$ si y sólo si $\forall s' : (s, s') \in R$ y $[\![\phi]\!]_{s'}^K = 1$

$[\![E\phi \text{ U } \psi]\!]_s^K = 1$ si y sólo si $\exists\,(s_0, s_1), \ldots (s_i, s_{i+1}) \in R : \exists i \geq 0$, $[\![\phi]\!]_{s_i}^K = 1$ y $\forall\, 0 \leq j \leq i\, [\![\phi]\!]_{s_j}^K = 1$

Operadores	
Simbolismo	Explicación
$X\phi$	Siguiente ϕ
$F\phi$	Eventualmente ϕ
$G\phi$	Globalmente ϕ
$A\phi$	Siempre ϕ
$E\phi$	Existe ϕ
ϕ U ψ	ϕ hasta ψ
ϕ R ψ	ϕ *release*ψ

Tabla 2.3. Simbolismo en operadores CTL

$[\![A\phi \text{ U } \psi]\!]_s^K = 1$ si y sólo si $\forall\,(s_0, s_1), \ldots (s_i, s_{i+1}) \in R : \exists i \geq 0$, $[\![\phi]\!]_{s_i}^K = 1$ y $\forall\, 0 \leq j \leq i\, [\![\phi]\!]_{s_j}^K = 1$

$[\![E\phi \text{ R } \psi]\!]_s^K = 1$ si y sólo si $\exists\,(s_0, s_1), \ldots (s_i, s_{i+1}) \in R : \forall i \geq 0$, $[\![\phi]\!]_{s_i}^K = 1$ o $\exists\, 0 \leq j \leq i\, [\![\phi]\!]_{s_j}^K = 1$

$[\![A\phi \text{ R } \psi]\!]_s^K = 1$ si y sólo si $\forall\,(s_0, s_1), \ldots (s_i, s_{i+1}) \in R : \forall i \geq 0$, $[\![\phi]\!]_{s_i}^K = 1$ o $\exists\, 0 \leq j \leq i\, [\![\phi]\!]_{s_j}^K = 1$

Estos se verán con ejemplos más adelante de una manera gráfica en la tabla (2.4).

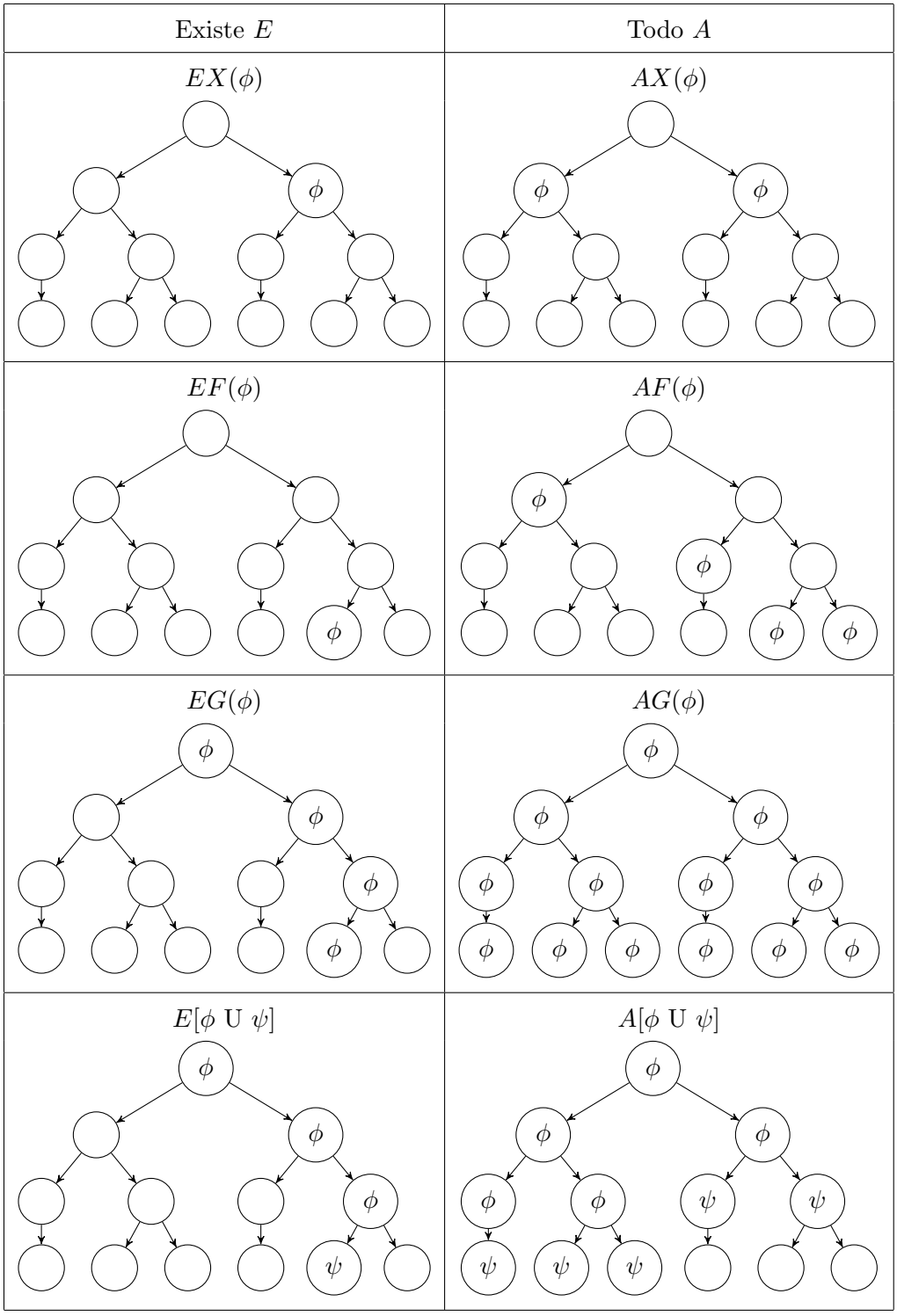


Tabla 2.4. Ejemplos representación de los operadores de CTL

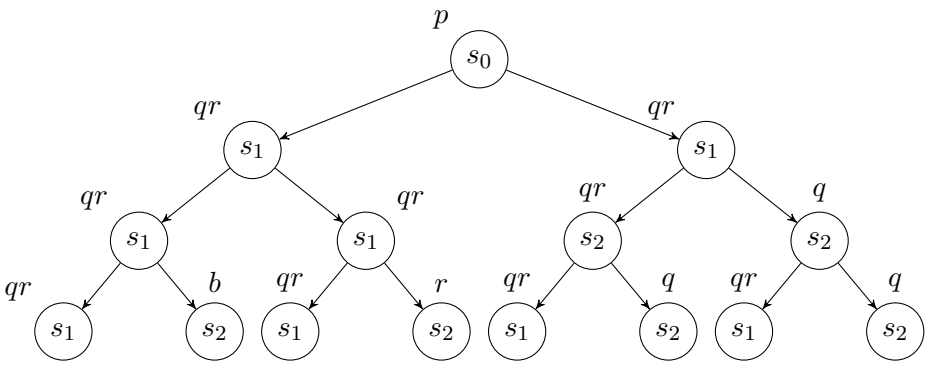


Figura 2.10. Representación gráfica del árbol para la fórmula CTL $EXp \wedge ArUq$

Ejemplo. Sea la Definición 2.3.3 se muestra una interpretación de las fórmulas CTL en K que cumple como la siguiente:

$$\phi = EXp \wedge ArUq$$

En la Figura 2.10 se observa el modelo en árbol de la estructura de la Figura 2.6 para la fórmula $EXp \wedge ArUq$ donde su interpretación representa que existe en algún momento q y que en todas las r está presente después el estado con q.

Un modelo puede ser satisfacible con diferentes soluciones correctas a una fórmula dependiendo de los operadores que use, esto significa que cumple en esos los caminos del árbol, o puede que no cumpla en ninguno esto dependera de la estructura.

De la misma manera se define la satisfactibilidad de una fórmula ϕ de CTL en K, cuando existe un modelo de ella, esto significa que después de analizarla bajo una interpretación es un modelo M de esta, por lo que es verdadera, o lo que es lo mismo $[\![\phi]\!]_S^K = 1$.

Por lo tanto siendo K nuestra estructura a usar, ϕ nuestra fórmula en CTL o TL, e I nuestros estados iniciales a verificar $[\![\phi]\!]_I^K = 0$ entonces $[\![\neg\phi]\!]_I^K = 1$ por lo que nuestra ruta π existe y si es el caso contrario no existe contraejemplo ni ruta. Un contraejemplo de $\phi = AFp$ debe mostrar que existe un camino infinito ρ tal que para todo estado de S se alcanza un ciclo con un estado con una proposición $\neg p$ es alcanzable en un algún nodo.

Ejemplo. Sea una estructura K:

- $S = \{s_0, s_1, s_2, s_3\}$.
- $I = \{s_0\}$.
- $R = \{(s_0, s_1), (s_0, s_2), (s_0, s_3), (s_1, s_1), , (s_2, s_2), (s_3, s_3)\}$.
- $L = \{(s_0, \{q\}), (s_1, \{p, q, r\}), (s_2, \{s, p\}), (s_3, \{p\})\}$.

Y siendo la fórmula $AF\,p$, esta es un modelo de K ya que$[\![AFp]\!]^K_{s_0} = 1$ porque cada camino de conduce a p, pero esta misma estructura no es un modelo de $AF\,s$ dado el contraejemplo con traza $\rho = s_0, s_1^\omega$ por lo que $C = (s_0, \{q\}), (s_1, \{p, q, r\})$ porque se queda en un ciclo donde no puede alcanzar s.

Con esto se muestra el poder computacional que se necesita tener para calcular árboles, en estructuras con ciclos.

2.4. Resumen

En esta sección se presentaron las bases de la Verificación de Modelos, definimos distintas lógicas modales como bases de la teoría de grafos, para usar conceptos que son clave en los siguientes puntos para la implementación y resolución del problema, por lo que se mostraron por su interrelación en algunos temas que se mezclan entre ambas.

Para las lógicas temporales sus modelos son una línea temporal y sobre ella realizan búsquedas alcanzables en el modelo, por ejemplo, buscar el valor siguiente en un nodo específico. Se vieron TL y CTL con el uso de estas lógicas se nos permite realizar búsquedas en un modelo de árbol o en los caminos del modelo, ambas son usada de manera distinta porque su unión es limitada al lenguage usado y es limitada a la herramienta a usar.

Así con el uso de la generación de contraejemplos de la herramienta de Verificación, al lograr la construcción de la estructura de Kripke y dar la representación de nuestra especificación con el uso de las fórmulas en la lógica temporal (CTL) buscaremos obtener una ruta satisfactoria para el problema de la planificación de trayectorias.

Capítulo 3

Planificación de trayectorias

En esta sección se presenta el problema de la Planificación de Trayectorias PT, planteada desde la introducción como: ¿Es posible generar trayectorias a través de la Verificación de Modelos? Para alcanzar este objetivo, se describe su planteamiento para la transformación del mapa a una estructura de Kripke, en términos de sus lugares, tomando como base la teoría mostrada en el capítulo anterior , la codificación de las especificaciones en requerimientos en fórmulas de lógica temporal CTL. Así para la resolución del problema se implementará un sistema prototipo, que a traves de su construcción con diferentes módulos y respectivos algoritmos, dará solución a la problemática. Con el sistema se selecciona la ubicación del mapa y los lugares de interés como entradas para la generación de la estructura y a partir de esta generar la fórmula en CTL. Por último, se muestra un ejemplo experimental del sistema con la construcción de una estructura basada en un mapa de Ciudad Universitaria.

3.1. Proceso para aplicar la Verificación de Modelos

Aunque la Verificación de Modelos ya fue definida en la sección anterior, también puede ser formulada como una técnica de verificación para explorar todos los estados de una estructura K, donde probando todas las posibles combinaciones, logra recorrer todos los caminos hasta dar con la solución de satisfactibilidad, por ello un verificador de modelos es una herramienta que examina todos los posibles escenarios de una manera sistemática y así se puede demostrar que existe un modelo que cumple con cierta propiedad (11).

El proceso para aplicar la VM sigue las siguientes fases:

- Generación de la estructura K y la fórmula ϕ
- Ejecucción del verificador
- Análisis de resultados

Para la generación de la estructura, se realiza la descripción del comportamiento de un sistema en una manera precisa, esta se realiza como una estructura de Kripke K y unas especificaciones usando lógica temporal dada una fórmula ϕ.

Para la ejecución se necesita que el verificador sea inicializado para que pueda establecer las diferentes opciones y directivas para realizar la verificación de manera exhaustiva, subsecuentemente la verificación se lleva a cabo de manera algorítmica para validar la satisfación del modelo bajo una fórmula o propiedad a verificar en todos los estados. En el análisis se tienen basicamente tres salidas, la propiedad cumple, la propiedad no cumple o el modelo es demasiado grande para los limites de la memoria de la computadora.

La Verificación de Modelos busca si existe un camino de estados o conjunto de caminos que cumplen ciertas características de un programa que, dada una cierta fórmula ϕ verifica si una estructura K es modelo de la fórmula, por esto la VM es una técnica efectiva para encontrar potenciales errores en un diseño.

Al buscar que una herramienta basada en VM verifique si una estructura es un modelo de una fórmula, internamente busca en cada una de sus ramificaciones de la estructura y el verificador puede encontrar si es o no es un modelo, dando un contraejemplo si no es un modelo de la estructura mostrando un camino. Este contraejemplo será la manera de afrontar nuestro problema de la PT y de la entrega de una solución. El contraejemplo mostrado por la herramienta será el primero encontrado dado su programación interna, esto parecera obtenido de manera aleatoria, pero demuestra que una estructura K para los estados iniciales establecidos, no satisface una fórmula ϕ y por lo que no es un modelo de ella, esto es de importancia por que el verificador busca la satisfactibilidad, cumpliéndose demostrar que sí es un modelo y en caso contrario, al no encontrar satisfacción a la fórmula entregará un contraejemplo, por lo que es de tomar a consideración que para la entrega de un camino se debe usar una fórmula inversa a las condiciones que se buscan para encontrar el contraejemplo, y esta será la forma de encontrar la ruta en nuestra estructura.

Cada uno de los conceptos y algoritmos que usa internamente el verificador de modelos lo veremos como una caja negra, ya que aún cuando se mostró en un inicio un diagrama de la herramienta, no se hace uso de sus componentes internos, por lo que los diseños de los sistemas de verificación, específicamente NuSMV y la forma que realiza la verificación para lograr la solución al modelo no se describe, pero puede verse en el artículo (12).

Así para la implementación se crea un prototipo que hace uso de un navegador y de un verificador de modelos, definiendo cuales son las especificaciones de entrada, salida y las otras restricciones o requerimientos. Con ello se verifica si la estructura es un modelo dados los requerimientos para el problema, y de ser así nos debe decir si lo es cumpliendo con los requerimientos para conseguir las rutas en nuestro mapa.

3.2. Planificación de Trayectorias usando Verificación de Modelos

Definición 3.2.1. (Planificación de trayectorias). El problema de planificación de trayectorias P_T se define como una tupla $P_T = \langle K_M, Req, T \rangle$ donde:

- K_M es el mapa elegido, que está descrito en términos de una estructura de Kripke (Definición 2.2.1).

- $Req = (L_i, L_f, Req_p, Req_o, Req)$ es el conjunto de todas los requerimientos tal que:

  L_i es el lugar de inicio y es un nodo en $S \in K_M$ tal que $L(I)$. Es la variable proposicional del lugar de inicio donde proposición tal que $I \in L(L_i)$.

  L_f es el lugar final, es un nodo en $S \in K_M$ tal que $L(Lf)$, y puede omitirse. Es la variable proposicional del lugar final donde proposición tal que $L(Lf)$.

  Req_p Requerimientos a pasar. Son los lugares que se requieren visitar y se definen como un conjunto posiblemente vacío de nodos $S \in K_M$ con proposiciones de AP donde $Req_p = \{p_1, p_2, \ldots, p_n\}$ tal que $p_i \in L(s_i)$

  Req_o Requerimientos a evitar u obstáculos. Son los lugares que NO se deben visitar y se define como un conjunto posiblemente vacío de nodos en $S \in K_M$ donde $Req_p = \{o_1, o_2, \ldots, o_n\}$ tal que $o_i \in L(s_i)$

  Req_t Restricciones secuenciales. Es un conjunto posiblemente vacío de requerimientos compuestas por expresiones de 3 tipos:

  Inmediata. $Req_{ti} : p \hookrightarrow q$ que significa que se requiere visitar primero p e inmediatamente después q.

  Secuencial. $Req_{ts} : p \multimap q$ que significa que se requiere visitar primero p y después en algún momento q.

Hasta. $Req_{th} : p -\upsilon\, q$ que significa que se requiere visitar p hasta cumplirse q.

- T es una trayectoria o ruta (secuencia de nodos en S) cuya traza satisface la función de traducción de los requerimientos en una fórmula de lógica temporal $f_\tau(Req) = \phi$, que se define más adelante y que será el camino (Definición 2.1.4), $\rho = s_0, s_1, s_2, \ldots s_i$, que puede ser un camino infinito dada la relación total en R o finito con final L_f, tal que la secuencia de pares $(s, L(s))$ satisface los requerimientos.

Cabe destacar que los requerimientos secuenciales pueden ser combinaciones de otros requerimientos secuenciales, y que también podemos tener requerimientos más complejos, a describirse en Req_m para obtener especificaciones más complejas, estas pueden ser directamente definidas con las operaciones de CTL y que pueden dar otras soluciones a diferentes planteamientos de la PT.

Sea para la $P_T = \langle K_M, Req, T\rangle$ el siguiente mapa ejemplo:

$K_M =$
$\{S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$
$I = \{s_1\}$
$AP = \{1, 2, 3, 4, 5, 6, m, b, s, e\}$
$R = \{(s_1, \{s_3\}), (s_2 : \{s_3, s_4, s_5, s_8, s_9\}), (s_3 : \{s_2\}),$
$\quad (s_4 : \{s_2, s_3, s_5, s_{11}\}), (s_5 : \{s_2, s_4, s_8, s_9\}),$
$L = \{(s_1, \{1, b, e, s\}), (s_2, \{2, m, s, e\}), (s_3, \{3, e, b\}), (s_4, \{4, s, e\}),$
$\quad (s_5, \{5, b\}), (s_6, \{6\})$
$\}$
$Req = \{Req_p = s_2, Req_o = s_4, Req_{ti} = s_2 -\star\, s_6\}$
$T = (s_1, \{1, b\}), (s_3, \{3\}), (s_2, \{2\}), (s_6, \{6\})$

Y su representación gráfica en la Figura 3.1:

Definición 3.2.2. (Función de traducción). $f_\tau : Req \mapsto \Phi$ Una función de traducción es una función que dadas unos requerimientos Req entonces entrega el conjunto de fórmulas en CTL de los requerimientos Φ.

Para la generación de la estructura K_M de acuerdo del mapa, es necesario conocer el tipo de mapa a usar, pues este define la manera de generar nuestra estructura para lograr el objetivo de la P_T por ello:

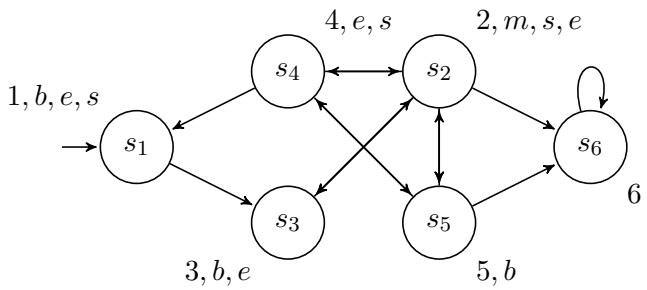


Figura 3.1. Ejemplo del problema de la P_T

Siendo K_M la estructura de Kripke sobre AP como una 4-tupla $K = \langle S, I, R, L \rangle$ a usar, AP en este caso siendo expresiones booleanas sobre variables y constantes para la identificación de lugares, tomaremos las proposiciones como información en nuestro mapa tal que serán el id de la ubicación e información relacionada al lugar para etiquetar los estados, palabras que puedan identificar nuestros lugares como :"estación", "metro", "restaurante", etc. Este etiquetado servira de base para generar las especificaciones a nuestras formulaciones para generar las trayectorias como lo veremos más adelante, entonces para el conjunto de proposiciones atómicas AP tomaremos en cuenta al conjunto de todos los lugares tales que el identificador del lugar sea único, por ello:

Definición 3.2.3. (Función identificador). $L^{id} : id \mapsto S$ Una función identificador es una función que dado un identificador único en AP entrega un nodo en S.

La función identificador es el estado o nodo donde la función es $L^{id}(id_i) = s_i$, además de la unión de las otras palabras clasificatorias del lugar, donde para $s \in S$, $L(s)$ contiene el conjunto de variables proposicionales que sean verdaderas en S, por lo que:

$$AP = palabras \cup id_S$$
$$L^{id}(id_i) = s_i$$

Con ello de la estructura K_M el conjunto de lugares S a utilizar, serán ubicaciones, cruces o lugares con los que identificaremos por identificadores(id) así que:

$$S = id_1, id_2, id_3, \ldots, id_i$$

Hay que tomar en consideración que se puede realizar una estructura mucho mayor e unificar mapas para hacerlo más completo pero también más complejo por la cantidad de estados.

Para el conjunto de lugares a iniciar, únicamente haremos uso de uno seleccionado por el usuario pero cabe destacar que solo tendremos una sola respuesta puesto que el

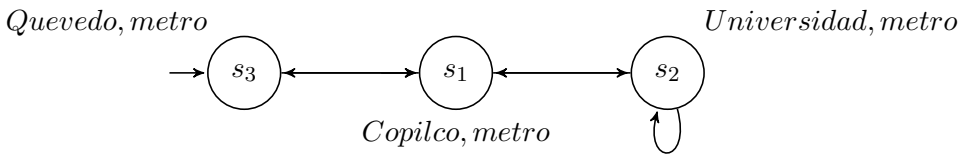


Figura 3.2. Representación ejemplo de 3 estaciones del metro en una estructura K

verificador puede ingresar varios estados iniciales pero sólo da una respuesta, por lo que:

$$I = id_{sel}$$

Esto obtiene tanto las etiquetas para AP con el conjunto de S, tanto los estados iniciales I que se tomaran de el conjunto S con una selección por el usuario. Para la relación de transición R entre nodos se tiene que definir de manera total para la herramienta, esto quiere decir que cada nodo debe tener por lo menos una transición y se podrá obtener verificando que un lugar tenga una transición y que obtendremos a partir de una matriz de adyacencia.

Con todas estas instancias tenemos la estructura K_M totalmente definida:

- S es el conjunto de todos los lugares a utilizar.
- $I \subseteq S$, es el conjunto de lugares iniciales .
- $R \subseteq S \times S$ es la relación de transición entre lugares tal que, $\forall\, s \in S, \exists\, t \in S : (s,t) \in R$.
- L es una función de etiquetado (o interpretación) $L : S \to 2^{AP}$.

Una representación de K_M de un mapa la rutas que tienen el metro donde cada estación tiene conexión con las siguientes más cercanas.

Como el ejemplo la representación de las últimas tres estaciones del metro con estaciones de Copilco, Universidad y M.A. de Quevedo como en la Figura 3.2 :

$S = \{s_1, s_2, s_3, \}$
$I = \{s_1\}$
$R = \{(s_1, s_2, s_3), (s_1, s_2), (s_2, s_1), (s_3, s_1)\}$
$L = \{(s_1, \{Copilco, metro\}), (s_2, \{Universidad, metro\}), (s_3, \{Quevedo, metro\})\}$
$AP = \{Copilco, Universidad, Quevedo, metro\}$

Para la obtención de una trayectoria, a partir de una especificación de requerimientos Req, se tiene que ser capaz de identificar una fórmula en lógica temporal ϕ, que cumpla

con las especificaciones deseadas, por eso a partir de estos requerimientos, que se traducen en una fórmula ϕ que se obtendrá a través de una conjunción de la función de traducción de cada uno de los requerimientos. Así la trayectoria que se obtiene verifica si es un modelo, en caso de que ϕ pueda satisfacerse.

Para lograr obtener T cada uno de estos requerimientos de Req debe escribirse de forma que pueda usarse dentro la lógica temporal CTL por lo que, a partir de la aplicación de la función de traducción en cada uno de los requerimientos tendremos:

- Requerimiento pase por todos los caminos Req_p
  Como los requerimientos $Req_p = p_1, p_2, \ldots, , p_i$ y la aplicación de la función de traducción tenemos:

$$f_\tau(Req_p) = EFp_1 \wedge EFp_2 \wedge \ldots \wedge EFp_i$$

- Requerimiento no pase por las obstáculos Req_o
  Como los requerimientos$Req_o = o_1, o_2, \ldots, , o_i$ y la aplicación de la función de traducción tenemos:

$$f_\tau(Req_o) = \neg(EFo_1 \vee EFo_2 \vee EFo_3 \ldots \vee EFo_i)$$

- Requerimiento de tiempo en actividades secuenciales Req_t(tendremos las inmediatas Req_{ti}, las no inmediatas Req_{ts} y las de hasta Req_{th}):

  Para Req_{ti} tendremos que para cada requerimiento secuencial en $Req_{ti} : p \hookrightarrow q$ entonces $f_\tau(p \hookrightarrow q) = (EF(p) \wedge EXq)$ por la conjunción de los requerimientos Req_{ti}es:

$$f_\tau(Req_{ti}) = EF(p_1 \wedge EXq_1) \wedge EF(p_2 \wedge EXq_2) \ldots EF(p_n \wedge EXq_n)$$

  Para Req_{ts} tendremos que para cada requerimiento secuencial $Req_{ts} : p -\!\star q$ entonces $f_\tau(p -\!\star q) = EF(p \wedge EFq)$ por la conjunción de los requerimientos Req_{ts} es:

$$f_\tau(Req_{ts}) = EF(p_1 \wedge EFq_1) \wedge EF(p_2 \wedge EFq_2) \ldots EF(p_n \wedge EFq_n)$$

  Para Req_{th} tendremos que para cada requerimiento secuencial $Req_{th} : p -\!\upsilon\, q$ entonces $f_\tau(p -\!\upsilon\, q) = f_\tau(Req_{th}) = p -\!\upsilon\, q = E(pUq)$ por lo que la conjunción de los requerimientos es Req_{th}:

$$f_\tau(Req_{th}) = E(p_1Uq_1) \wedge E(p_2Uq_2) \ldots \wedge E(p_nUq_n)$$

  Por eso finalmente la función de traducción en los requerimientos secuenciales es la conjunción de los diferentes tipos de requerimientos secuenciales

$$f_\tau(Req_t) = f_\tau(Req_{ti}) \wedge f_\tau(Req_{ts} \wedge f_\tau(Req_{th})$$

- Existen otro tipo de requerimientos que se pueden formular en conjuntos de fórmulas de CTL usando todos los operadores dentro de esas lógicas por lo que su formulación y traducción se realiza directamente en una fórmula en CTL.

  $$Req_m = \phi$$

  El uso de Req_m es opcional para utilizar otras fórmulas, que pueden ayudar a crear otro tipo de requerimientos más complejos, que el usuario puede crear si tiene conocimiento de la lógica temporal, para ello es necesario que la fórmula cumpla con las características de CTL para incorporar el uso de los otros operadores de la lógica temporal como: $\phi = EGp$.

Al formular la especificación a nuestra ruta será una conjunción de todos los requerimientos de esta manera:

$$f_\tau(Req) = f_\tau(Req_p) \wedge f_\tau(Req_o) \wedge f_\tau(Req_t) \wedge f_\tau(Req_m) \tag{3.1}$$

Y con ello obtenemos la fórmula a verificar en la estructura $f_\tau(Req) = \phi$.

La respuesta podrá ser el conjunto vació $\emptyset$ en T, en caso de que no exista la trayectoria que satisfaga los requerimientos. Por eso el problema de P_T se satisface cuando la trayectoria T no es vacía, esto quiere decir que estos requerimientos deben poder validar el modelo y si se satisface el mapa definido es un modelo de la fórmula.

Con la verificación de modelos podemos visualizar si la estructura es un modelo de la fórmula propuesta, con lo cual solamente tendremos una respuesta de veracidad o falsedad, por lo que para la obtención de una trayectoria o ruta T será necesario conseguir una prueba, para la aplicación de la técnica contamos con el contraejemplo. Para obtenerlo se realiza una negación a la fórmula con los requerimientos tal que $\phi = \neg Req$, ya que no es posible conseguir testigos o caminos de la búsqueda cuando la especificación en CTL es verdadera, así la herramienta hará uso de contraejemplos para mostrar la falsedad de la especificación con una cantidad de pasos que muestra la ruta a tomar. Por lo que de la Definición 3.2, será:

$$\phi = \neg Req = \neg(Req_p \wedge Req_o \wedge Req_{ti} \wedge Req_{tn} \wedge \ldots Req_m) \tag{3.2}$$

Así la trayectoria será la respuesta de la herramienta en un camino de estados.

Con esta información se presenta la siguiente estructura ejemplo, esta es $K_{ej} = \langle S, I, R, L \rangle$ donde:

$S = \{s_1, s_2, s_3, s_4, s_5, s_7, s_8, s_9, s_{10}, s_{11}\}$
$I = \{s_1\}, s_{10}$
$AP = \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, m, b, s, e\}$

$R = \{(s_1 : \{s_3\}), (s_2 : \{s_3, s_4, s_5, s_8, s_9\}), (s_3 : \{s_2\}),$
$(s_4 : \{s_2, s_3, s_5, s_{11}\}), (s_5 : \{s_2, s_4, s_8, s_9\}),$
$(s_7 : \{s_1, s_{10}\}), (s_8 : \{s_2, s_5, s_9\}),$
$(s_9 : \{s_2, s_5, s_8\}), (s_{10} : \{s_3, s_7\}), (s_{11} : \{s_{11}\})\}$
$L = \{(s_1, \{1, b, e, s\}),$
$(s_2, \{2, m, s, e\}),$
$(s_3, \{3, e, b\}),$
$(s_4, \{4, s, e\}),$
$(s_5, \{5, b\}),$
$(s_7, \{7, b\}),$
$(s_8, \{8\}),$
$(s_9, \{9, b\}),$
$(s_{10}, \{10, m\}),$
$(s_{11}, \{11\})$

Y su representación gráfica 3.3:

Y sean Req con los siguientes requerimientos:

- Pasos $Req_p = 9, 4, 3$

$$f_\tau(Req_p) = EF9 \wedge EF4 \wedge EF3$$

  Significa pasar por los lugares 9, 4, 3.

- Obstrucciones $Req_o = 5$

$$f_\tau(Req_o) = \neg(EF5)$$

  Significa evitar el lugar 5.

- Actividades secuencial $Req_t = 3 \hookrightarrow 2$

$$f_\tau(Req_t) = EF(3 \wedge EX2)$$

  Significa que cuando eventualmente pase por 3 inmediatamente después pase a 2.

  Entonces como $Req = Req_p \wedge Req_o \wedge Req_t$ la fórmula de los requerimientos será:

$$Req = (EF9 \wedge EF4 \wedge EF3) \wedge (\neg(EF5)) \wedge EF(3 \wedge EX2)$$

Lo que significa pasar por 9, 4 y 3, evitar 5 y cuando eventualmente pase por 3 inmediatamente pasa 2, así la trayectoria obtenida será:
$T = (s_1, \{1, b, e, s\}), (s_3, \{3, b, e\}), ((s_2, \{2, m, s, e\}), (s_4, \{4, e, s\}), (s_9, , \{9, b\}), (s_8, \{8\}))^\omega$

Dado que la relación es total, será parte de la implementación el decidir donde parar la ruta ya sea dando Lf o cuando comience el ciclo. Por lo que para este problema ya tenemos la respuesta a $P_T = \langle K_M, Req, T \rangle$

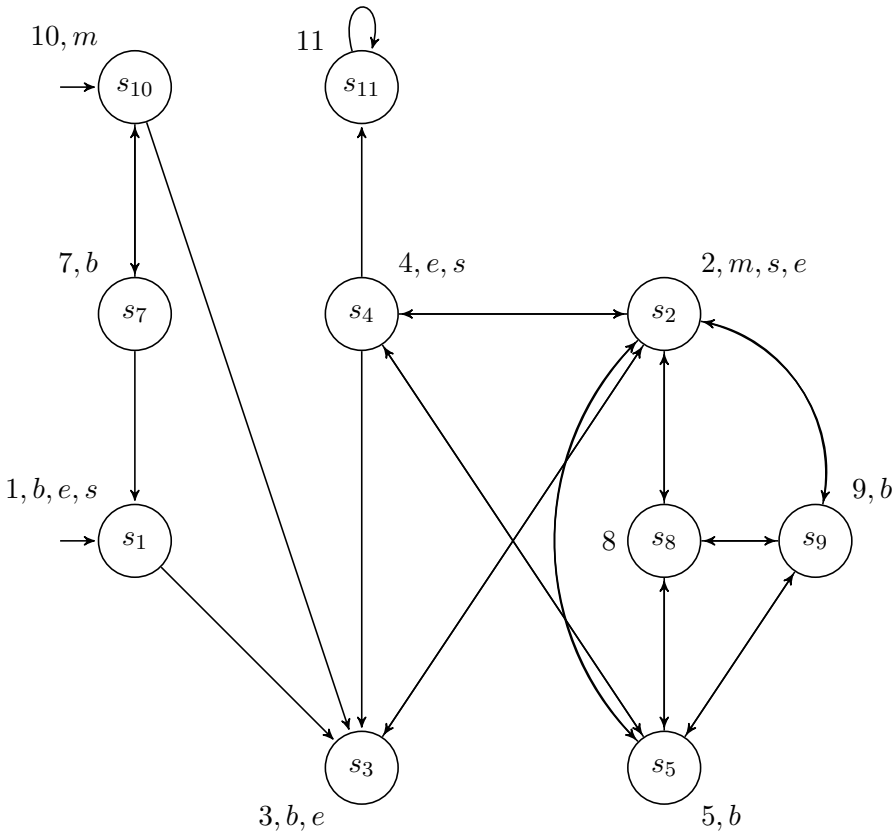


Figura 3.3. Mapa ejemplo para la P_T

Otros ejemplos de trayectorias obtenidas con requerimientos, su traducción a operaciones CTL para K_{ej} y de la respuesta de la herramienta se pueden ver en la Tabla 3.1:

Esto muestra la capacidad del planteamiento para obtener trayectorias dependiendo de los requerimientos impuestos.

3.3. Implementación

En esta sección se presenta la implementación para la generación de trayectorias con base en las especificaciones generadas con el uso de CTL, se describe su formulación para uso en la herramienta NuSMV, la creación del contraejemplo en la herramienta cuando una especificación cumple, así hace entrega de una sentencia de verdadero, y en caso de no cumplirse una sentencia de falso y entrega un contraejemplo, mostrado con el prototipo, que cuando es formulada correctamente entregará la trayectoria deseada para

Req_t	Traducción a $\phi = \neg$ CTL	Trayectoria ρ de T
$Req_p = \{9, 4\}, Req_o = \{5\}$	$\neg((EF(9) \wedge EF(4)) \wedge \neg(EF(5))$	$s_{10}, s_3, s_2, s_4, (s_9, s_8)^\omega$
$Req_p = \{9, 4\}, Req_o = \{m\}$	$\neg(E(mUb) \wedge EFm)$	$(s_{10}, s_7)^\omega$
$Req_p = \{11\}$	$\neg EF(11)$	$s_{10}, s_3, s_2, s_4, (s_{11})^\omega$
$Req = s -\upsilon 11$	$\neg E(s\,U(11))$	$\emptyset$
$Req = s -\upsilon 11$	$\neg E(e\,U11)$	$s_1, s_3, s_2, s_4, (s_{11})^\omega$
$Req = m -\upsilon b -\upsilon s -\upsilon 11$	$\neg E(E(E(EFm\,U\,b)Us)U11)$	$s_{10}, s_3, s_2, s_4, (s_{11})^\omega$
$Req = m -\upsilon e$	$\neg E(mUe)$	$s_{10}, s_3, (s_2, s_5)^\omega$

Tabla 3.1. Rutas contraejemplos de P_T con requerimientos y traducción

su visualización, también se mostraran los pasos y algoritmos para el uso del sistema, además de la experimentación de un mapa.

Después de definidas las necesidades para la generación de estructuras K_M y la tupla del problema de la P_T se propone una interfaz en un navegador web para la implementación del sistema diseñada en html y javascript con el uso de la API de Google Maps para la visualización del mapa y de los lugares de interés, seleccionados desde el mismo sistema, los cuales se presentaran como una forma de etiquetado al modelo y con ellos se hará uso de la herramienta en NuSMV para la respuesta a la especificación, fórmula descrita en LTL o CTL, para encontrar una ruta.

Para la implementación del prototipo se programaran los siguientes módulos y se mostrará el pseudocódigo y código en la traducción de los mismos.

Algoritmo para la creación de prototipo con sus funcionalidades:

- Generación de la estructura K_M

  Palabras a mapear

  Coordenadas a estados, puntos obtenidos con etiquetado

  Conversion de rutas en una función de transición

- Programa de traducción
- Toma de requerimientos
- Formulación de ϕ
- Verificación del modelo

- Obtención de trayectoria
- Visualización de la ruta

Para ello se tiene un análisis de como obtener la información, los lugares a añadir a la estructura, así como el etiquetado para definir rutas más complejas através de la verificación de modelos. Cada uno de estos puntos se realiza para finalmente unificar los módulos para la generación del prototipo.

3.3.1. Información y datos para la estructura de Kripke

La información, datos y estructuras para el algoritmo que generará la creación de la estructura de Kripke, se basaran en una estructura de datos con la información de la Ubicación con la siguiente forma para su uso en la herramienta de VM tanto para el manejo en su visualización con la siguiente estructura para cada lugar:

Ubicación
id:String
nombre:String
etiqueta:String
localización:longitud y latitud
marcador:obj
listaAdyacencia:Arr

Tabla 3.2. Estructura de objeto Ubicación

Por lo que se generará un arreglo de ubicaciones 3.2 que tiene la información necesaria para la programación de los algoritmos, teniendo el id, nombre, localización de la Ubicación dada por el API, la etiqueta será la palabra *et* utilizada para encontrar la ubicación, el marcador es para la visualización en el navegador y la lista de adyacencia 2.1.8 contendrá los lugares adyacentes al nodo, esto para la facilitar la construcción y modificación de la estructura en el sistema.

En nuestro prototipo para la obtención de lugares se crea una función a partir de una entrada de palabras *et* ingresadas por el usuario, relacionadas a la definición de lugares, estas palabras también nos serviran como parte de las proposiciones atómicas AP, esto dado un radio de búsqueda en una ubicación (longitud y latitud), en el prototipo definido como un radio de 1500 m y la función de nearbySearch() es directa del API (13) y recibe como parametro una consulta req, esta es una localización, con un radio y una palabra a buscar y entrega un arreglo de lugares con su información bajo esa consulta, además del estatus de la consulta en status como OK, si fue correcta y push(resultado) hace la

función de añadir al arreglo el resultado.

Entrada: Palabras et
Resultado: Lugares:Conjunto de S

```
initialization
Arreglo ubicaciones ⟵ ∅
Arreglo req[et] ⟵ ∅
for i ∈ et do
    req[i]=(loc centro del mapa, radius:1500, name:et[i])
end
for i ∈ et do
    nearbySearch(req[i])
    for j ∈ result do
        if nearbySearch(req[i]).status = "OK" then
            ubicaciones ⟵ push(result)
        end
    end
end
```

Algoritmo 1: Obtención lugares S

La información obtenida se guarda en una lista de tablas de Ubicación 3.2. Con ello se guarda en una lista visible al usuario tal que se muestre en la pantalla como la Figura 3.4. Si no es de interés tener uno de los lugares agregados por el algoritmo, se podrá eliminar ese lugar de la lista mediante el uso de un botón en el navegador.

Para la creación de la relación de transición R, de nodos en S se hizo uso de una matriz de adyacencia MA (Definición 2.1.7), para almacenar las transiciones, esto tiene sus ventajes en contra de la lista de adyacencia, que se usa de manera inversa para obtener los puntos a partir del código de NuSMV en caso de guardar el mapa que pero tuvo sus particularidades. Para la traducción a NUSMV se eligió la matriz de adyacencia puesto que es la forma de como entrega las distancias el API de Google pero este podría modificar si quisieramos hacerlo más eficiente.

$$MA = R = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix} \tag{3.3}$$

Esta función de transición de igual manera se obtiene a través de los servicios de GoogleMaps con su API, para la búsqueda de cada una de las ubicaciones y la relación con otras en un radio, eligiendo palabras relacionadas al lugar a buscar y etiquetar. Esto se guardará en la MA que representa la función. Para la implementación se tiene un radio seleccionado de búsqueda de 200m.

Figura 3.4. Vista gráfica de la lista de lugares en el prototipo

Algoritmo de creación para la función de transición:

Entrada: Palabras et
Resultado: MA (Matriz de incidencia)

```
initialization
Arreglo ubicacionescer ⟵ ∅
Arreglo req[et] ⟵ ∅
for i ∈ S do
    for j ∈ et do
        req[(i*et.length)+j] ⟵(loc:ubi[i], radius:200, name:et[j])
        if nearbySearch(req).status = "OK" then
            for k ∈ result do
                req[(i*et.length)+j] ⟵ (loc:ubi[i], radius:200, name:et[j])
                if nearbySearch(req).status = "OK" then
                    ubicacionescer[i] ⟵ push(result)
                end
            end
        end
    end
end
for i ∈ S do
    for j ∈ S do
        for k ∈ ubicacionescer do
            req[(i*et.length)+j]=loc:ubi[i], radius:200, name:et[j]
            if nearbySearch(req) = "OK" then
                MA[i, j] ⟵ ubicacionescer[i, k]
            end
            else
                MA[i, j] ⟵ 0
            end
        end
    end
end
```

Algoritmo 2: Algoritmo para la función de transición MA

Todo esto se verá en el prototipo a partir de botones y cajas de texto interactivas en lado derecho del prototipo visto como en el Anexo E.1.

A partir de R tenemos las relaciones entre nodos y por la información de Ubicaciones, dadas por et, tenemos el etiquetado en cada nodo de S por lo que ya tenemos la función de etiquetado L guardada en la lista de Ubicaciones 3.2. Para las proposiciones se hará uso de los id y de información importante que sea de nuestro interés, en este caso se hará uso de las palabras relacionadas al lugar, que podría incluir otros datos pero en este modelado tendremos esta información.

Por lo que al final el modelo $K_M = \langle S, I, R, L \rangle$ se quedará con esta estructura:

- $S = id_1, id_2, id_3 \ldots, id_i$
- $I = id_{sel}$, refiriendose al inicio seleccionado
- $R = MA$
- AP =palabras $\cup id_s$ en L^{id}
- L es una función de etiquetado (o interpretación) $L : S \to 2^{AP}$.

Para nuestro caso:

$$K_M = \left\langle \{id_1, id_2, id_3 \ldots, id_i\}, \{id_{sel}\}, MA, S \to 2^{AP} \right\rangle \tag{3.4}$$

Esta ecuación será la base para la escritura de la estructura en la herramienta, así para el prototipo se hizo uso una nueva función de traducción de los datos tomados con javascript para su uso con la herramienta de NuSMV y su uso en la misma mostrada en el anexo A.

Cada uno de los requerimientos debe ser escrito en CTL para usarse en una fórmula, para que la herramienta sea capaz de verificar la veracidad de la especificación o ser capaz de dar un contraejemplo.

3.3.2. Generación de la fórmula con los requerimientos

Para la generación de la fórmula final Req a usar se implementará la siguiente ecuación:

$$Req = Req_p \wedge Req_o \wedge Req_t \wedge \ldots Req_m \tag{3.5}$$

Dada la ecuación en la implementación, es únicamente tarea del usuario seleccionar el estado inicial I, los obstáculos Req_o y los lugares a pasar Req_p. Para Req_t seleccionando la casilla "Ordenado", realizará Req_{ts} en términos de secuencia y si se desea inmediatez Req_{ti} se seleccionará "Inmediatez". Para la restricción manual Req_m, el usuario debe que escribir la fórmula dentro de la caja de texto para fórmulas complejas, tomando en cuenta la selección de la lógica a usar como se puede ver en la Figura 3.5.

La función de traducción f_τ se realiza al tener la estructura K_M y su implementación para la obtención con los requerimientos mostrados, se puede ver en el anexo A, hay que considerar que debe incluirse en el código de la fórmula, el nombre que se asignó a los estados para su uso. Por lo que al negar la función de traducción, podremos verificarla en la estructura y así nos entrega el contraejemplo para la visualización en el sistema por lo que $[\![f_\tau(Req)]\!]_I^{K_M} = 1$ es un modelo y $\neg f_\tau(Req)$ nos entregará el contrajemplo.

Figura 3.5. Vista para ingresar requerimientos

Así para el uso con NuSMV se necesita ingresar en el lenguaje de la herramienta, la especificación deseada que al final se realiza de esta manera: "SPEC ϕ" tomando en cuenta que $\neg$ se traduce como !, $\wedge$ como & , $\vee$ como $\|$, $\Diamond$ como F, X como X, $\Box$ como G; tomando en cuenta que siempre se debe usar el operador cuantificador A o E, si se usa un operador temporal todos los operadores disponibles se encuentran en el manual (1).

Con ejemplos traducidos a la herramienta de NuSMV, tenemos las fórmulas de las sección anterior y la tabla 3.4 que son:

$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_10, s_11\}$
$I = \{1\}$
$R = \{(s_1 : \{s_3\}), (s_2 : \{s_3, s_4, s_5, s_6, s_8, s_9\}), (s_3 : \{s_2, s_6\}), (s_4 : \{s_2, s_3, s_5, s_6, s_{11}\}),$
$(s_5 : \{s_2, s_4, s_6, s_8, s_9\}), (s_6 : \{s_2, s_3, s_5, s_8, s_9\}), (s_7 : \{s_1, s_10\}), (s_8 : \{s_2, s_5, s_9\}),$
$(s_9 : \{s_2, s_5, s_8\}), (s_{10} : \{s_3, s_7\}), (s_{11} : \{s_{11}\})\}$
$L = \{(s_1, \{1, bi, em, es\}),$
$(s_2, \{2, met, es, em\}),$
$(s_3, \{3, em, bi\}),$
$(s_4, \{4, es, em\}),$
$(s_5, \{5, bi\}),$

Operadores en lógica temporal	
Simbolismo	Traducción
$\neg\phi$	$!\phi$
$F\phi$	$F(\phi)$
$G\phi$	$G(\phi)$
$X\phi$	$X(\phi)$
$\phi_1 \wedge \phi_2$	$\phi_1 \& \phi_2$
$\phi_1 \vee \phi_2$	$\phi_1 \mid \phi_2$
$\phi_1 U \phi_2$	$[\phi_1 U \phi_2]$
$\phi_1 R \phi_2$	$[\phi_1 V \phi_2]$

Tabla 3.3. Traducción de operadores lógica temporal a NuSMV

Fórmula en LTL	Comando en NuSMV
$\phi = \neg((EF(9) \wedge EF(4)) \wedge \neg(EF(5))$	`SPEC !((EF(loc=9)&EF(loc=4))&!(EF(loc=5)));`
$\phi = \neg((biEUmet) \wedge EFmet)$	`SPEC !(E[bi U met]& EF met);`
$\phi = \neg EF(11)$	`SPEC ! F loc=11;`
$\phi = \neg(esEU(11))$	`SPEC !E[es U loc=11];`

Tabla 3.4. Traducción de contraejemplos con operadores CTL

$(s_6, \{6\}),$
$(s_7, \{7, bi\}),$
$(s_8, \{8\}),$
$(s_9, \{9, bi\}),$
$(s_{10}, \{10, met\}),$
$(s_{11}, \{11\})\}$
$AP = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, met, bi, es, em\}$

Las respuestas a los caminos generados pueden cumplir nuestro objetivo de manera contra intuitiva a lo esperado, pero son validadas por la herramienta, lo que puede indicar que la formulación a la especificación y requerimientos se pueden haber hecho de manera errónea. Otra consideración de la Definición 2.3.4 al tener una relación total en la estructura, siempre genera un camino con un ciclo infinito, pero que habrá resuelto la fórmula que satisface la especificación y ya será tarea del algoritmo de visualización

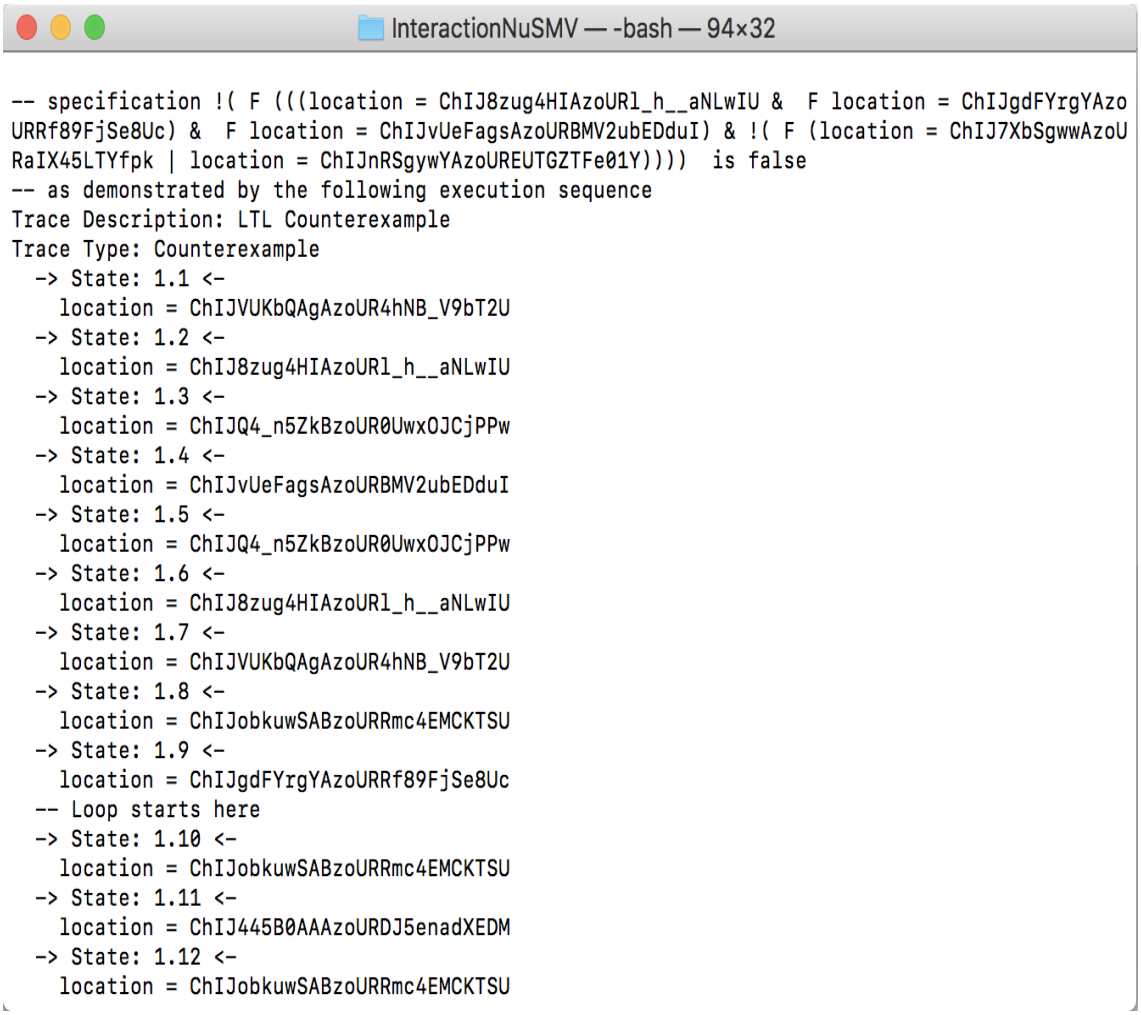


Figura 3.6. Respuesta a la especificación en NuSMV

definir hasta donde entregará la ruta deseada si no existe L_f.

Existen dos respuestas de la especificación en NuSMV la primera será la de si la especificación cumple, esto quiere decir que la fórmula es infinitamente frecuente verdadera, donde sólo otorga un verdadero y la segunda respuesta es cuando no cumple la fórmula, que entregará el contraejemplo, por ello la negación mostrada en la ecuación 3.2.

La respuesta de NuSMV se confiere desde la terminal del sistema y se verá como la Figura 3.6.

Esta respuesta nos entrega el camino en el árbol temporal, que satisface la fórmula y con eso la trayectoria deseada, la respuesta es la secuencia de estados $\rho = s_1, s_2, ..., s_n$ que es infinita por la relación R y muestra los cambios en cada una de las etiquetas, esta

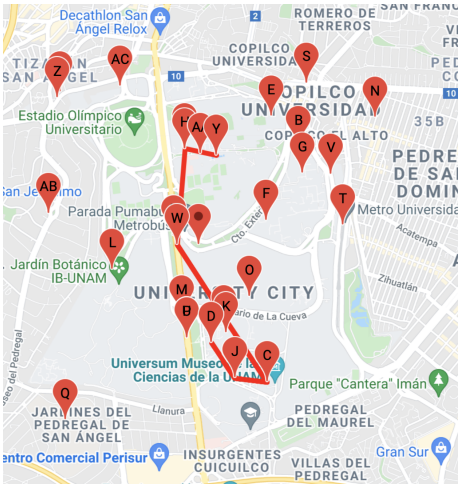


Figura 3.7. Visualización de respuesta en el prototipo

respuesta la rediccionaremos a un archivo de texto para su procesamiento en el prototipo con una función de lectura para su guardado en un arreglo de puntos y así mostrar la visualización de la trayectoria en el navegador como en la Figura 3.7.

3.4. Experimentación

Con el prototipo de sistema se propone para la experimentación obtener como entrega resultado, las trayectorias en el conjunto específico de puntos dados en el mapa de Ciudad Universitaria.

Se implementa una versión del prototipo con uso de NuSMV 2.6.0 en el lenguaje de programación Javascript implementando la tecnología del API de GoogleMaps para visualización del mapa y los lugares de interés. Para las pruebas de implementación, se utilizó una computadora con las siguientes características: Sistema operativo Mac Os Monterey, Procesador Intel i7 2.8 GHz., 16 GB de RAM. Y con el navegador Safari 13.1.12 y Brave V1.32.113 que cambio su forma gráfica pero no su usabilidad.

El prototipo incorpora el uso de cuatro módulos en general:

- La página de visualización, la cual despliega la información ingresada por el usuario para visualizar de manera gráfica el mapa y los lugares a elegir, programada en javascript y que hace unión a los demás módulos, generando la estructura de Kripke, las especificaciones, algoritmos de entrada, salida, de visualización y la función de traducción a la herramienta.
- La API de GoogleMaps, la cual es nuestra herramienta para obtener la información

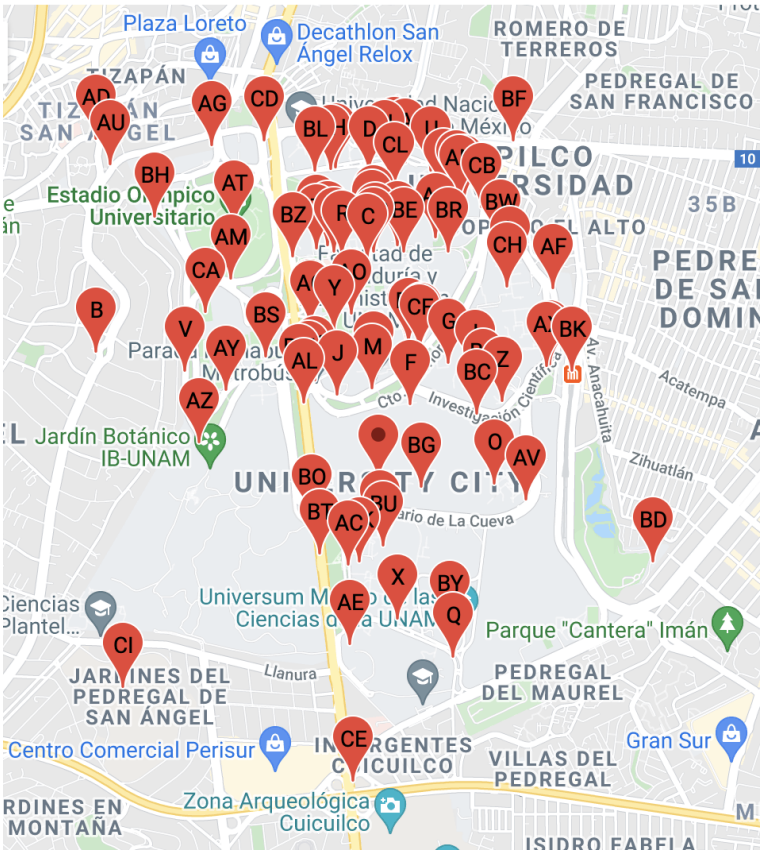


Figura 3.8. Mapa y lugares para ejemplo en CU

y para desplegar un mapa gráfico.

- El servidor o programa encargado en correr la herramienta de NuSMV.
- La herramienta de NuSMV, que es la que realizará los pasos para obtener la ruta con las especificaciones.

Ejemplo en Ciudad Universitaria, sea nuestro mapa el de la Figura 3.8. Para una visualización sencilla en el presente documento, se intercambia el identificador del lugar por letras del alfabeto como se muestra en el la tabla del anexo C. Y la estructura sea la estructura K_{ej} mostrada en el apéndice D.

Nuestra estructura ejemplo presenta 91 nodos y 228 aristas. Con esto a partir del mapa K_M, formularemos las siguientes requerimientos para visualizar el resultado en una trayectoria satisfactoria esta fórmula Req será:

Para estado inicial tendremos de inicio Metro Copilco

$$I = L^{id}(CC)$$

Siendo los pasos obligatorios el Universum M, el MUCA N, MUAC BA y el Estado Olimpico R:

$$f_\tau(Req_p) = EFM \wedge EFN \wedge EFBA \wedge EFR$$

las obstáculos, el Museo de Arq AS y metro Universidad CH:

$$f_\tau(Req_o) = \neg EF(AS \vee CH)$$

y la secuencia de Pabellon Nacional de la Biodiversidad AW y Universum M :

$$f_\tau(Req_t) = EF(EFAW \wedge EXM)$$

Por lo que:

$$f_\tau(Req) = Req_p \wedge Req_o \wedge Req_t \tag{3.6}$$

Y finalmente:

$$f_\tau(Req) = (EFM \wedge EFN \wedge EFBA \wedge R) \wedge \neg EF(AS \vee CH) \wedge EF(EFAW \wedge EXM) \tag{3.7}$$

Que significa se obtendra la trayectoria empezando por metro Copilco como CC, pasando el Universum como M, el MUCA como N, el MUAC como BA y el Estado Olimpico como R, evite Museo de Arq. como AS y metro Universidad como CH y consiga la secuencia obligatoria de Pabellon Nacional de la Biodiversidad AW y Universum como M en el mapa K_M es el camino con pasos: $T = CC, Z, BF$ como Copilco, Derecho, Filosofía y su visualización se muestra en la terminal como en la Figura 3.9, con un tiempo de obtención del sistema en 0.038 s.

Con ello se realiza la obtención de resultados en el mapa que se visualizara en el prototipo como en la Figura 3.10.

Se muestra la tabla con tiempos de respuesta dado el mismo mapa K_M, para obtención de diferentes fórmulas con distintas trayectorias 3.5.Este tiempo es la suma de los tiempos de la inicialización de NuSMV, la colocación de memoria, el proceso interno y la entrega de los resultados. El proceso de inicialización por si sólo toma 0.012 s.

Figura 3.9. Respuesta en NuSMV

Fórmula	Tiempo de Respuesta
$\phi = \neg(E[EFestacionUlocation = M]$	0.030 s
$\phi = \neg EF(location = BR \wedge EFlocation = CD)$	0.037 s
$\phi = \neg(EF(location = AQ))\wedge!(EF(location = BO))$	0.024s
$\phi = \neg EF(metrobus)$	0.021 s
$\phi = \neg(E[(EF\neg facultad)U(EGfacultad)])$	0.021 s

Tabla 3.5. Tiempos de respuesta para diferentes fórmulas

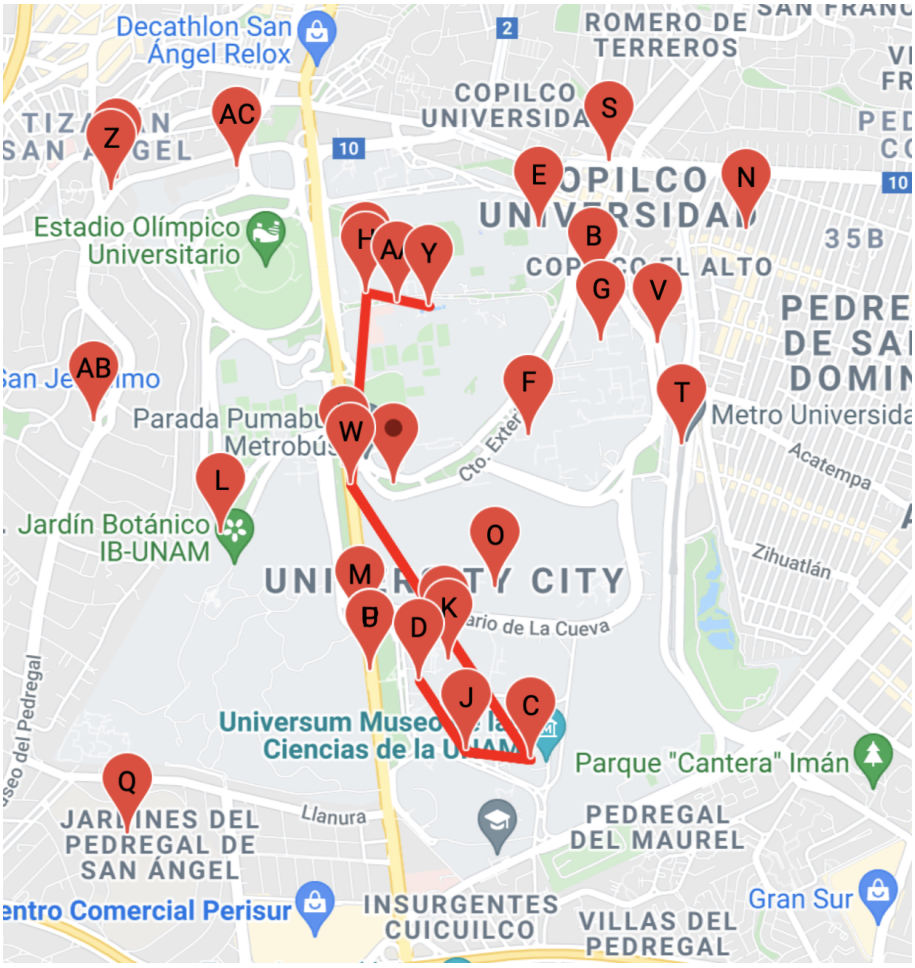


Figura 3.10. Visualización de la trayectoria en el mapa

3.5. Conclusiones

En esta sección se presentó la próblematica de la Planificación de trayectorias basada en el uso de Verificación de Modelos con las fórmulas de lógicas temporales, la implementación y resolución del problema.

La Verificación de Modelos puede modelar muchos problemas que están muy optimizados, cabe destacar que se deben de entender las diferencias teorícas y prácticas de este acercamiento, para generar un modelo abstracto haciendo uso de una estructura de Kripke, con ciertas propiedades a cumplir, que pueden variar del estado inicial al estado final.

Con la construcción de la estructura de Kripke y de las fórmulas en la lógica temporal (CTL) y su uso de Verificación de Modelos en este caso concreto de NuSMV, se

logró la obtención de una trayectoria dadas unas especificaciones para la planificación de trayectorias cumpliendo el objetivo de este documento, el cual puede extenderse a diferentes usos y que sólo muestra una fracción de las aplicaciones que podrían resultar haciendo uso de estas lógicas, para lograr una aplicación se debe tomar en cuenta que entre más cercana la información al mundo real, tales que muestren información verídica a lo que acontece en la realidad, como podrían ser los mapas de recorrido del metro, metrobus, rutas de camiones, entre otras, con esto se pueden realizar diferentes aplicaciones a las mostradas para obtener una trayectoria pero siguiendo esta forma de solucionar el problema de la PT.

Como puede verse en el prototipo y su implementación, el uso de esta técnica muestran su factibilidad para obtener trayectorias, donde la Verificación de Modelos podría aplicarse para diferentes situaciones de la vida cotidiana.

Capítulo 4

Conclusiones

En esta tesis se describió la aplicación de la Verificación de Modelos para demostrar que es posible usar esta técnica para la planificación de trayectorias, así mostrar la factibilidad de su uso para aplicarse como una técnica para el desarrollo de nuevas aplicaciones, como se fue demostrado en los anteriores procedimientos.

Para lograr la aplicación se expuso la creación y discretización de la estructura P_T con sus conjuntos, para ello fue esencial definir y contextualizar el uso que se quiso dar y la escritura de las especificaciones en una fórmula lógica para obtener una trayectoria que cumpliera con las especificaciones deseadas, como lo expuesto en la implementación de la aplicación, donde se pudo cumplir de una manera sencilla y amigable al usuario con algunas especificaciones y pruebas en el sistema. Se presentó el uso de lógica temporal para lograr algunas especificaciones, de manera distinta se mostró como para lograr especificaciones más complejas deben de escribirse directamente en lógica temporal, en este caso CTL y esto requiere conocimiento de la lógica temporal para usar en cada verificador de modelos, como es el caso de aquí en NuSMV, pero que la vuelve propensa a encontrar nuevos usos, si se es capaz de definir la especificación de la fórmula y el etiquetado correspondiente a la estructura. Por ello si se logra esa capacidad podría darse diferentes usos como es el planificar un viaje, el crear rutas en caso de emergencias y mejora en toma de decisiones para visitar lugares de interés, hasta en la planificación para la logística en diferentes industrias teniendo en cuenta diferentes especificaciones a formular que pueden lograr la planificación de trayectorias.

El objetivo general de la tesis se alcanzó de forma positiva a través del cumplimiento de los objetivos específicos. En el primer objetivo específico para desarrollar la abstracción de un mapa de propósito general a una estructura, se estudiaron y definieron las bases de la lógica modal en especial de la lógica temporal, para esto se estudiaron las sintaxis y semántica relevantes para la creación del prototipo, este análisis se realizó en el segundo capítulo. En el segundo objetivo específico de codificar rutas en un mapa de propósito general en términos de un lenguaje lógico, con las bases expuestas en el segundo capítulo y con ello se implementó una herramienta mostrada en el tercer capítulo para

la transcripción de la entrada dada por un usuario a la comunicación con la herramienta para dar un contraejemplo con la trayectoria que cumple la especificación en una lógica temporal. El tercer objetivo específico se desarrolló un prototipo con interfaz gráfica para generar trayectorias a partir de la visualización de un mapa integrando una herramienta de Verificación de Modelos, lo cual se muestra en los algoritmos y el sistema presentado para lograr visualizar una trayectoria en un mapa para cumplir con la especificación.

De esta manera se dio solución al planteamiento de la Planificación de Trayectorias usando Verificación de Modelos y se dieron algunos ejemplos de como llevar a cabo esta tarea.

4.1. Perspectivas de investigación y aplicaciones

Con lo presentado en el trabajo se podría continuar con esta investigación tanto para la comparación de la eficiencia con otros verificadores (25), como la forma de comunicación con el sistema, esto dependera de cuanto sea el tiempo tomado, como la forma de hacerlo más amigable al usuario.

Para la implementación en un software comercial o de uso cotidiano se deben de hacer diferentes cambios tanto para la obtención del datos a partir de la API de GoogleMaps, pues es necesaria la comunicación con los servidores de Google para los datos, esto en la programación de la obtención de la información, causa un conflicto por la forma de entrega de información y los límites encontrados para la obtención de la MA y de los lugares. Por lo tanto para obtener la respuesta y trayectoria son modificaciones manuales a la estructura, puesto que la API usada al ser gratuita está restringida en lo que Google Maps puede entregar, especialmente a las peticiones de información la cuál esta restringida a cierto número máximo de elementos, por lo que se tendría que adaptar para que la estructura no tenga que ser modificada en caso de no tener cierta información en la relación total.

Otra avance relacionado al sistema actual, sería mejorar el algoritmo de creación de la estructura de una manera real o el modelado completo del mapa, esto podría representar un nuevo tipo de problema a solucionar pero que es limitante por la forma de uso del API, así para que el espacio de los nodos siga un mapa continuo tal como en sistemas de autonavegación (22), aún necesitan obtenerse los lugares de un mapa o de un acercamiento a la navegación de multimodelo (26), para tener una estructura conexa.

Ya que la funcionalidad del prototipo es limitada en la construcción de la estructura, esto se lograría solucionar con cambios en los algoritmos propuestos para la construcción del mapa en una estructura y debería buscarse una alternativa a la obtención de la relación de transición, puesto que por la forma uso de API de Google y las restricciones a su respuesta, muchas veces pueden dejar nodos sin transiciones y que limitan el uso

del prototipo al realizar el ingreso manual de lugares con sus etiquetas y sus relaciones con otros nodos, por lo que para amplia de su uso, esto tendría que darse como en otras aplicaciones, para la generación de la estructura como en aplicaciones con retroalimentación de usuarios para que fuera una tarea sencilla de realizar o modificar los algoritmos para siempre tener al menos una transición a otro estado.

Con base en lo encontrado a través de las herramientas utilizadas que fueron la API de GoogleMaps, Javascript y NuSMV se podría hacer una selección diferente a la elegida para obtener parametros similares, haciendo uso de diferentes herramientas disponible con Google (13), podría incluirse un servicio de Geolocalización, tráfico para un uso en tiempo real, para hacer con ellos una selección más amplia de rutas según nuestro criterio y llegar a puntos o resultados interesantes para otro tipo de aplicaciones.

Por otra parte este trabajo podría combinarse para el funcionamiento con nuevas aplicaciones tanto en sistemas automáticos de planeación, como sistemas autónomos con navegación (22), haciendo uso de especificaciones complejas, lo que incluiría una funcionalidad innovativa a usos industriales y comerciales hasta como en sustentabilidad (27). Existen muchos tipos de trabajos que buscan solucionar el problema de la planificación de trayectorias de diferentes enfoques, inclusive para solucionar problemas existentes como el de preferencias de manejo (28) para lograr un nuevo sistema, por ello el uso de la Verificación de Modelos podría ampliar el rango de uso, claro está, adaptando la forma de expresar las nuevas especificaciones que sera la parte compleja a realizar para aplicar la técnica.

Se tiene en consideración que el uso de otras lógicas con mayor expresividad, como cálculo-μ o lógicas multimodales (10), que podrían usarse en combinación con otros algoritmos, cuya aplicación podría utilizarse a futuro pero que en la actualidad la vuelven difícil de implementar, ya que no se cuentan con verificadores en estas lógicas y porque se necesitan nuevas herramientas que logren un tiempo de respuesta aceptable al usuario, por lo que es posible que en un futuro la generación de nuevas trayectorias, usando distintas estructuras e incluso nuevas formas de especificación, se logre haciendo uso de lógicas más complejas para nuevos objetivos pero que se necesita avance en estas áreas.

Apéndice A

Programa de traduccion

```
async function transformaANUSMV(ids,matrix){
//Reseteo de palabras
var primero=1;
obstaculos=[];
inicios=[];
pasos=[];

let obs = getDatosLista('listaobstaculos');
let inis = getDatosLista('listainicio');
let pas = getDatosLista('listaporpasar');
for(let ind1 of obs){
 obstaculos.push(lugares.find(lugares => lugares.name ==ind1.value).id);
}
for(let ind2 of inis){
 inicios.push(lugares.find(lugares => lugares.name ==ind2.value).id);
}
for(let ind3 of pas){
 pasos.push(lugares.find(lugares => lugares.name ==ind3.value).id);
}
let miCheckbox = document.getElementById('cbox');
```

```
if (verificaConexionMatriz(matrix)!=0){
 alert("Existen puntos sin transicion agregar manualmente");
}

var texto="";
texto += "MODULE main\n";
texto += "\tVAR\n";
//variables para pruebas en CTL
texto += "\t\tlocation:{";



for(let i=0; i<ids.length;i++)
{
 if(i==ids.length-1){
  texto += ids[i]+"};\n";
 }
 else{
  texto += ids[i]+",";
 }
}
texto += "\tASSIGN\n";
//A revisar para inicios
if(inicios.length==0){
 texto += "\n";
}
else if(inicios.length==1){
 texto += "\t\tinit(location):="+inicios[0]+";\n";
}
else{
```

```
 texto += "\t\tinit(location):={";
 for(let i=0; i<iinicios.length;i++)
 {
 if(i==inicios.length-1){
 texto += inicios[i]+"};\n";
 }
 else{
 texto += inicios[i]+",";
 }
 }
}
texto += "\t\tnext(location):=\n";
texto += "\t\t\tcase\n";

//Asignacion de las transiciones XXXXXXXX
for(let i=0; i<ids.length;i++)
{
 primero=1;
 texto += "\t\t\t\tlocation = " +ids[i]+":{";
 for(let j=0; j<ids.length;j++)
 {
 //A revisar como verificar ultimo para agregar coma o no
 if((matrix[i][j]!=0) && (i!=j))
 { if(primero==1){
 texto += ids[j];
 primero=0;
 }
 else{
 texto +="," +ids[j] ;
```

```
 }
 }
 }
 texto += "};\n"
 if(i==ids.length-1){
 texto += "\t\t\tesac;\n";
 }
}
// Codigo para usar DEFINE y otro servira para definir lugares o
//Diferencias para verificar ubicaciones
texto += "\tDEFINE\n";
for(let i=0; i<palabras.length;i++){
 texto += "\t\t"+palabras[i]+":=";
 for(let j=0; j<lugares.length;j++){
 if(lugares[j].etiqueta==palabras[i]){
 texto +=" location = "+lugares[j].id+" |";
 }
 }
 if(texto.endsWith('|')){
 texto = texto.slice(0, -1)+";\n";
 }else{
 texto +=  ";\n";
 }
}
//Para agregar obstaculos y pasos
if(obstaculos.length!=0 & pasos.length!=0){
//texto += "CTLSPEC !(";
//Inicio de lugares de pasos  XXXXXXXX
 if(miCheckbox.checked){
```

```
for(let i=0; i<pasos.length;i++){
if(i==0 & pasos.length==1){//primero y uno
texto += "CTLSPEC ! (EF(location="+pasos[i]+")) ";
}
else if(i==0 & pasos.length!=1){//primero
  texto += "CTLSPEC ! (EF(location="+pasos[i]+") &  ";
}
else if(i==pasos.length-1){//ultimo
lugares.find(lugares => lugares.id ==pasos[i]).location;
texto += "EF location="+pasos[i]+")".repeat(pasos.length-1);
}
else{//en medio
texto += "EF (location="+pasos[i]+"& ";
}
}
 }else{
for(let i=0; i<pasos.length;i++){
if(i==0 & pasos.length==1){//primero y uno
texto +="CTLSPEC !(EF(location="+pasos[i]+")) ) \n";
}
else if(i==0 & pasos.length!=1){//primero
texto += "CTLSPEC !EF ((EF location="+pasos[i]+" & ";
}
else if(i==pasos.length-1){//ultimo
lugares.find(lugares => lugares.id ==pasos[i]).location;
texto += "EF location="+pasos[i]+")";
//texto += "EF location="+pasos[i]+")".repeat(pasos.length-1);
}
else{//en medio
```

```
texto += "EF location="+pasos[i]+" & ";
}
}
}
texto += "&";
//Inicio de obstaculos ^
for(let i=0; i<obstaculos.length;i++)
{
if(i==0 & obstaculos.length==1){//primero y uno
texto += "!(EF(location="+obstaculos[i]+"))) ";
}
else if(i==0 & obstaculos.length!=1){//primero
//texto += "CTLSPEC !(EEF(location="+obstaculos[i]+")) & ";
texto += "!(EF (location="+obstaculos[i]+"|";
}
else if(i==obstaculos.length-1){ //ultimo
lugares.find(lugares => lugares.id ==obstaculos[i]).location;
texto += "location="+obstaculos[i]+")))";
}
//
else{ //en medio
texto += "location="+obstaculos[i]+"| ";
}
}
//Cuando no hay una ruta
}else if(obstaculos.length==0 & pasos.length==0){
alert("No hay especificacion");
}else{
//Cuando solo obstaculos ^
```

```
//texto += "CTLSPEC !("
if(obstaculos.length!=0){
for(let i=0; i<obstaculos.length;i++)
{
if(i==0 & obstaculos.length==1){//primero y uno
texto += "CTLSPEC (EF(location="+obstaculos[i]+")) ";
}
else if(i==0 & obstaculos.length!=1){//primero
//texto += "CTLSPEC !(EEF(location="+obstaculos[i]+")) & ";
texto += "CTLSPEC (EF (location="+obstaculos[i]+"|";
}
else if(i==obstaculos.length-1){ //ultimo
lugares.find(lugares => lugares.id ==obstaculos[i]).location;
texto += "location="+obstaculos[i]+"))";
}
//
else{ //en medio
texto += "location="+obstaculos[i]+"| ";
}
}
}
//Cuando solo pasos  X
else{
//sin orden de pasos  ^
if(miCheckbox.checked == false){
for(let i=0; i<pasos.length;i++){
if(i==0 & pasos.length==1){//primero y uno
texto += "CTLSPEC !(EF(location="+pasos[i]+")) ";
}
```

```
 else if(i==0 & pasos.length!=1){//primero
 texto += "CTLSPEC !(EF(location="+pasos[i]+") & ";
 }
 else if(i==pasos.length-1){//ultimo
 lugares.find(lugares => lugares.id ==pasos[i]).location;
 texto += "EF(location="+pasos[i]+"))";
 }
 else{//en medio
 texto += "EF(location="+pasos[i]+") & ";
 }
 }

 // con orden de pasos
 }else{
  for(let i=0; i<pasos.length;i++){
if(i==0 & pasos.length==1){//primero y uno
texto += "CTLSPEC ! (EF(location="+pasos[i]+")) ";
}
else if(i==0 & pasos.length!=1){//primero

texto += "CTLSPEC ! (EF(location="+pasos[i]+") &  ";
}
else if(i==pasos.length-1){//ultimo
lugares.find(lugares => lugares.id ==pasos[i]).location;
texto += "EF location="+pasos[i]+")".repeat(pasos.length-1);
}
else{//en medio
texto += "EF(location="+pasos[i]+"& ";
}
```

```
  }
 }
 }
}

//Para agregar ubicaciones por pasar
document.getElementById("codigotext").value=texto;
console.log("Se genero text para script");

}
```

Apéndice B

Ejemplificación de lugares


```
[
  {
    "name": "Universum",
    "place_id": "ChIJ8zug4HIAzoURl_h__aNLwIU",
    "location": "{ "lat ": 19.3114346, "lng ": -99.1805749 }",
    "etiqueta": "museo"
  },
  {
    "name": "University Museum Contemporary Art",
    "place_id": "ChIJvUeFagsAzoURBMV2ubEDduI",
    "location": "{ "lat ": 19.3147106, "lng ": -99.18540349999999 }",
    "etiqueta": "museo"
  },
  {
    "name": "Museum of Pathological Anatomy",
    "place_id": "ChIJ8OaeXRsAzoURV-aOyjZ18AQ",
    "location": "{ "lat ": 19.3309, "lng ": -99.177765 }",
    "etiqueta": "museo"
  },
  {
```

```
  "name": "MUSEO ARQ UNAM",
  "place_id": "ChIJobkuwSABzoURRmc4EMCKTSU",
  "location": "{ "lat ": 19.330655, "lng ": -99.18772299999999 }",
  "etiqueta": "museo"
},
{
  "name": "Morphology Museum",
  "place_id": "ChIJE-ZmFxsAzoURe2apy1Kwteo",
  "location": "{ "lat ": 19.3286924, "lng ": -99.177407 }",
  "etiqueta": "museo"
},
{
  "name": "MUCA CU - Museo Universitario de Ciencias y Arte",
  "place_id": "ChIJ445B0AAAzoURDJ5enadXEDM",
  "location": "{ "lat ": 19.3311362, "lng ": -99.1877231 }",
  "etiqueta": "museo"
},
{
  "name": "Museo de Zoología Alfonso L. Herrera",
  "place_id": "ChIJi9rjzHEBzoURchPPhblYHgI",
  "location": "{ "lat ": 19.324844, "lng ": -99.1806548 }",
  "etiqueta": "museo"
},
{
  "name": "Zona Arqueológica, Zacatepetl",
  "place_id": "ChIJe64E9tz_zYURGRn7zzvS3h8",
  "location": "{ "lat ": 19.3082926, "lng ": -99.1982361 }",
  "etiqueta": "museo"
},
```

```
{
  "name": "Paseo de las Esculturas (Reserva Ecológica del Pedregal de San Ángel)",
  "place_id": "ChIJ1Z3rbwwAzoUR-Kkgk9K9Qf8",
  "location": "{ "lat ": 19.3160638, "lng ": -99.1843376 }",
  "etiqueta": "museo"
},
{
  "name": "Museo de Anatomía",
  "place_id": "ChIJ45yoed4BzoUREgW-G1ItXHc",
  "location": "{ "lat ": 19.3333913, "lng ": -99.1802355 }",
  "etiqueta": "museo"
},
{
  "name": "\Variante de Kepler" de Manuel Felguérez",
  "place_id": "ChIJ7XbSgwwAzoURaIX45LTYfpk",
  "location": "{ "lat ": 19.3155402, "lng ": -99.18418009999999 }",
  "etiqueta": "museo"
},
{
  "name": "Pabellón Nacional de la Biodiversidad",
  "place_id": "ChIJQ4_n5ZkBzoUR0UwxOJCjPPw",
  "location": "{ "lat ": 19.3117894, "lng ": -99.18337869999999 }",
  "etiqueta": "museo"
},
{
  "name": "Vestigios arqueológicos de Cuicuilco",
  "place_id": "ChIJDezCVjMBzoURsWbUSt7cVlo",
  "location": "{ "lat ": 19.3035195, "lng ": -99.18572999999999 }",
  "etiqueta": "museo"
```

```
},
{
  "name": "Universidad",
  "place_id": "ChIJz5LRsRAAzoURQHcXyJCKa7M",
  "location": "{ "lat ": 19.324405, "lng ": -99.173913 }",
  "etiqueta": "estacion"
},
{
  "name": "Ciudad Universitaria",
  "place_id": "ChIJVUKbQAgAzoUR4hNB_V9bT2U",
  "location": "{ "lat ": 19.3227897, "lng ": -99.1884637 }",
  "etiqueta": "estacion"
},
{
  "name": "Centro Cultural Universitario",
  "place_id": "ChIJ68HI4AoAzoUREOsheFAkwYE",
  "location": "{ "lat ": 19.3151515, "lng ": -99.1875385 }",
  "etiqueta": "estacion"
},
{
  "name": "Olympic Pool University",
  "place_id": "ChIJgdFYrgYAzoURRf89FjSe8Uc",
  "location": "{ "lat ": 19.3301942, "lng ": -99.1850158 }",
  "etiqueta": "alberca"
},
{
  "name": "Alberca Fernando Martí Haik",
  "place_id": "ChIJJVKjFxUAzoURZaEJ1KOHNG0",
  "location": "{ "lat ": 19.3147361, "lng ": -99.1695877 }",
```

```
    "etiqueta": "alberca"
  },
  {
    "name": "Olympic Pool Gym UNAM",
    "place_id": "ChIJnRSgywYAzoUREUTGZTFe01Y",
    "location": "{ "lat ": 19.3303608, "lng ": -99.1863518 }",
    "etiqueta": "alberca"
  },
   {
    "name": "Herbario Nacional de México",
    "place_id": "ChIJ8zbqkn__zYURd62W-dMgsZI",
    "location": "{ "lat ": 19.3208175, "lng ": -99.194132 }",
    "etiqueta": "herbario"
  }
]
```

Apéndice C

Tabla de lugares y su simbología

Id	Lugar	Etiqueta	Simbolo
ChIJ1cw4PQIAzoURAUofmOez1lQ	P.Parada del pumabús Economía	pumabus	A
ChIJFc2llvr_zYURQ059_FCFrU8	School Swimming Club Canada	alberca	AA
ChIJgdFYrgYAzoURRf89FjSe8Uc	Olympic Pool University	alberca	AB
ChIJhbEB5f__0YUR8pLOUUjyd7s	UNAM Faculty of Law	facultad	AC
ChIJHQsvmwUAzoURQkI0w123rjQ	P.Parada Pumabús Anexo de Ingeniería	pumabus	AD
ChIJHRoVEkMBzoURv0WgfxQ-C-A	Posgrado Facultad de Contaduria y Administración	facultad	AE
ChIJi9rjzHEBzoURchPPhblYHgI	Museo de Zoología Alfonso L. Herrera	museo	AF
ChIJiaaZeggAzoUREEQD6DI0WOc	P.Metro CU	pumabus	AG
ChIJjyiW3Q8AzoURjdMDW5uvlhA	UNAM - Facultad De Ciencias	facultad	AH
ChIJk-DTjAgAzoUREf8Vc4UET10	P.Trabajo Social	pumabus	AI
ChIJK1_l7vgBzoURWhX_6W2h6zw	Edificio D - Facultad de Ingeniería	facultad	AJ
ChIJKR9ztxwAzoURnLdu1phqTTY	Faculty of Veterinary Medicine UNAM	facultad	AK
ChIJKXuXtggAzoURyvmp9WPrifM	P.Facultad de Contaduria Y Administracion	pumabus	AL
ChIJlex9kgEAzoUR3xn3m_PTY54	P.Parada del pumabús Derecho	pumabus	AM
ChIJm5MTIzEBzoURERcgw6a4MXo	Facultad de Ciencias Políticas y Sociales UNAM	facultad	AN
ChIJMfe6mxAAzoURi3kcHhoKZp0	P.Pumabus Ruta 2	pumabus	AO
ChIJn68yXHIAzoURaMK37JFmAJo	P.Parada PUMA BUS Archivo General	pumabus	AP
ChIJnRSgywYAzoUREUTGZTFe01Y	Olympic Pool Gym UNAM	alberca	AQ
ChIJo28xGwMAzoURHDp7b-5jXBk	Bicipuma Medicina	pumabici	AR
ChIJobkuwSABzoURRmc4EMCKTSU	MUSEO ARQ UNAM	museo	AS
ChIJoyHY9wIAzoURViRV0ktmFQ0	Faculty of Dentistry	facultad	AT
ChIJP3Bzrfz_zYUR9xKQXlV9PMA	P.Pumitas	pumabus	AU
ChIJp5LJJAMAzoURIZ_rKuM4afg	FACULTAD DE MEDICINA	facultad	AV
ChIJQ4_n5ZkBzoUR0UwxOJCjPPw	Pabellón Nacional de la Biodiversidad	museo	AW
ChIJQxSqTgYAzoURs9NLVsbu2dY	Roberto stadium "Tapatio"Mendez	estadio	AX
ChIJr_tcERAAzoURR8nD6NkNess	Faculty of Chemistry Set D - E	facultad	AY
ChIJrQvKHwcAzoURcnflz02_2P8	Facultad de Química UNAM	facultad	AZ
ChIJ1Z3rbwwAzoUR-Kkgk9K9Qf8	Paseo de las Esculturas	museo	B
ChIJSaG0FQsAzoURbiHKYdLd40Q	P.Centro Cultural Universitario / Muac	pumabus	BA
ChIJSzTqjwcA0oURDJ7PufncnzI	Estadio Pedregal	estadio	BB
ChIJT7Ey7HQAzoURt_JoOk4JVXQ	División de Estudios de Posgrado de Economía	facultad	BC
ChIJtUe_thsAzoURMIPSqWVnAsg	P.Parada Pumabús CENDI	pumabus	BD
ChIJuRL_nj7_zYURRK5UePl20zE	ElParque Wellness Center	alberca	BE
ChIJv2jzsgEAzoURBfJVQm4yR_M	Facultad de Filosofía y Letras UNAM	facultad	BF
ChIJVTrGowgAzoURomdeLtHRAuM	Facultad de Contaduría y Administración UNAM	facultad	BG
ChIJvUas1AYAzoURll5XtpiCHs8	P.Facultad de Arquitectura	pumabus	BH
ChIJvUeFagsAzoURBMV2ubEDduI	University Museum Contemporary Art	museo	BI
ChIJVUKbQAgAzoUR4hNB_V9bT2U	Ciudad Universitaria	estacion	BJ
ChIJvxfsy_3_zYUR5XAvtNilKDA	P.E7	pumabus	BK

ChIJwRJV0xwAzoURDucBpvxrHgw	Facultad de Medicina UNAM	facultad	BL
ChIJXVonXgYAzoURx36yK0x1r78	Estadio Manuel Neri Fernandez (Field 2)	estadio	BM
ChIJXwVHVQEAzoURGI48gptVtgk	P.Facultad de Ingenieria	pumabus	BN
ChIJy5pLuwcAzoURG14Up4B1uPw	Estadio de Practicas	estadio	BO
ChIJyxi4LwEAzoURMSsBBZN1IKg	Faculty of Architecture	facultad	BP
ChIJz5LRsRAAzoURQHcXyJCKa7M	Universidad	estacion	BQ
ChIJz9X9XP7_zYUR4gSAZe8hcrg	Estadio Olímpico Universitario	estadio	BR
ChIJZf1pZfj_zYURX_n_4ithyx4	Albercas Delfín	alberca	BS
ChIJ_cDkSRIAzoUR4Ufu6ENlOAI	P. Facultad de Ciencias Políticas y Sociales UNAM	pumabus	BT
ChIJ_ezy8wEAzoURWi_C-7ZT-gw	Parada Derecho	pumabus	BU
ChIJ0yHimhAAzoUR1wCsCTR4Qac	Parada Ruta 4	pumabus	BV
ChIJ78V_J-P_zYURO0TfPI6n0nY	Campos de Futbol I		BW
ChIJ8zbqkn__zYURd62W-dMgsZI	Herbario Nacional de México		BX
ChIJb4u7IBAAzoUR6jzmNA2lhD4	Talleres Facultad de Ciencias		BY
ChIJeTS9NZQBzoURvqleq3-MIlE	Edificio K- Facultad de Ingenieria		BZ
ChIJ2Qf01A8AzoURA_gn7Cz-HlA	P.Facultad de Ciencias	pumabus	C
ChIJJVKjFxUAzoURZaEJ1KOHNG0	Alberca Fernando Martí Haik		CA
ChIJmaw6CwQAzoURSW0SLRZvhfM	Parada Frontones	pumabus	CB
ChIJpWKqeh0AzoURRMzbXEEHcBE	Metro Copilco	estación	CC
ChIJrcalMQwAzoURyXCMkV9Xjk8	Espacio Escultótico UNAM		CD
ChIJS1rNN_n_zYURStPTZQQvbX4	Anexo Facultad de Filosofía		CE
ChIJzT3VSAAAzoURnpb1m98vl6U	Parada Filosofía	pumabus	CF
ChIJccpu-woAzoUR_pLLg38Lmmk	Metrobus Zona Cultural	metrobus	CG
ChIJz5LRsRAAzoURQHcXyJCKa7M	Metro Universidad	estacion	CH
ChIJv5KZSAAAzoURDctwzleCyD0	Facultad de Filosofía y Letras	facultad	CI
ChIJWWEKEwgAzoURITRDHs2RXHQ	Estacion de metrobus CU	metrobus	CJ
ChIJXZ5_CgQAzoURQht2IV5_3RM	Parada Pumabus CELE	pumabus	CK
ChIJL1qfRgoAzoURp0oxVkj9hqE	Glorieta Coche		CL
ChIJ445B0AAAzoURDJ5enadXEDM	MUCA CU - Museo Universitario de Ciencias y Arte	museo	D
ChIJ45yoed4BzoUREgW-G1ItXHc	Museo de Anatomía	museo	E
ChIJ52TgnAQAzoURj-ilEzIFI2I	P.Pumabus IIMAS	pumabus	F
ChIJ53R5Wv3_zYURt6wjelMZp4c	Estadio De Beisbol CU	estadio	G
ChIJ68HI4AoAzoUREOsheFAkwYE	Centro Cultural Universitario	estacion	H
ChIJ7XbSgwwAzoURaIX45LTYfpk	"Variante de Kepler" de Manuel Felguérez	museo	I
ChIJ7Xp_uAcAzoURhjoacZO_Tnc	Parada del pumabús Estadio de Prácticas	estadio	J
ChIJ80aeXRsAzoURV-aOyjZ18AQ	Museum of Pathological Anatomy	museo	K
ChIJ88OikwUAzoUR-GOTzs6j5t4	Facultad de Ingeniería UNAM - Ciencias Básicas	facultad	L
ChIJ8zug4HIAzoURl_h__aNLwIU	Universum	museo	M
ChIJa0Z6TgcAzoURvemGoaU1Sfw	P.Parada del pumabús MUCA	pumabus	N
ChIJa4PYXvz_zYURI7BQSjl3tOY	Campo 1	estadio	O
ChIJb3dUyxwAzoURrjE0zpKq5QI	P.Parada del pumabús Medicina	pumabus	P
ChIJb53YJgIAzoUR6ihxLRp_r3c	FACULTY OF ECONOMICS Building B	facultad	Q
ChIJbS_A9___zYURqtHFIz5ss08	Cd. Universitaria - Estadio C.u.	estadio	R
ChIJDezCVjMBzoURsWbUSt7cVlo	Vestigios arqueológicos de Cuicuilco	museo	S
ChIJDWjKmgUAzoURnshUVQDyQio	Annex School Of Engineering Div. Basic Sciences	facultad	T
ChIJDzptqwYAzoURhtg7PtNEHDY	P.Ingenieria	pumabus	U
ChIJE-ZmFxsAzoURe2apy1Kwteo	Morphology Museum	museo	V
ChIJe64E9tz_zYURGRn7zzvS3h8	Zona Arqueológica, Zacatepetl	museo	W
ChIJE6fgWwEAzoURcEMkqIEWwT8	Faculty of Engineering, UNAM	facultad	X
ChIJE6jRbQgAzoUR7bydI0VfKA0	P.Parada del "Pumabus", camiones de CU	pumabus	Y
ChIJEX2MLAIAzoUR7esq9asRAmY	Edificio C - Facultad de Derecho	facultad	Z

Tabla C.1. Tabla de lugares, ids y símbolos a usar

Apéndice D

Representación simbólica de CU en Estructura de Kripke

S={A,AA,AB,AC,AD,AE,AF,AH,AI,AJ,AK,AL,AM,AN,AO,AP,AQ,AS,AT,AU,AV,AW,AZ,
B,BC,BD,BE,BF,BG,BI,BJ,BL,BP,BQ,BR,BT,BU,BV,BW,BX,BY,BZ,CA,CB,CC,CD,CE,
CF,CG,CH,CI,CJ,CK,CL,D,E,F,H,I,J,K,L,M,N,P,S,T,V,W,X,Y,Z}

I={CC}

R={(A:{P,CF,BU}),
(AA:{AL}),
(AB:{AH,BG,L,T,BY,BZ}),
(AC:{V}),
(AD:{M}),
(AE:{L,T,AE,BY,BZ}),
(AF:{J,Y}),
(AH:{BI,B,I,CG,H}),
(AI:{AP,BJ}),
(AJ:{P,CK,F}),
(AK:{AK}),
(AL:{AL}),

(AM:{AM}),

(AN:{BC,AN}),

(AO:{D,AB,AQ}),

(AP:{X,Z,AC,AV,AZ,AJ,BL}),

(AQ:{BW}),

(AS:{Z,AZ,AT,BL}),

(AT:{E}),

(AU:{W,M,BI,I}),

(AV:{BI,B,I}),

(AW:{AP}),

(AZ:{BR}),

(B:{BI,I}),

(BC:{CF,BU}),

(BD:{L,T,AE}),

(BE:{B,I,AW,H,H}),

(BF:{D,AS,BX,H,BQ,BC}),

(BG:{BJ}),

(BI:{CF,F,CB}),

(BJ:{BF,X,Z,AC,AJ}),

(BL:{BJ}),

(BP:{BF,X,Z,AC,AJ}),

(BQ:{BJ}),

(BR:{BF}),

(BT:{AN}),

(BU:{A,CF,CI}),

(BV:{AO,BD}),

(BW:{AU}),

(BX:{BJ}),

(BY:{AH,L,T,AE,BZ}),

(BZ:{E,AS,D,AQ}),

(CA:{BF,X,Z,AC,AZ,AT,BP}),

(CB:{AS,D,AB}),

(CC:{X,Z,AC,AV,AT,AJ,BL}),

(CD:{AO}),

(CE:{X,Z,AC,AJ,BP,CF,BU}),

(CF:{A,CK,CI,BU}),

(CG:{AW}),

(CH:{BJ}),

(CI:{Z,AV,AZ,AT}),

(CJ:{A,CK,CI,BU}),

(CK:{CF,F,CB}),

(CL:{BD}),

(D:{AS,AB,AQ}),

(E:{K,AB,CC}),

(F:{P,CK,CB}),

(H:{BI,B,I,AW,H}),

(I:{BI,B,AW}),

(J:{N,AI,Y}),

(K:{V,E}),

(L:{AH,BG,T,AE,BY}),

(M:{AW,BJ}),

(N:{J}),

(P:{A,F,CB}),

(S:{K,AF}),

(T:{AW}),

(V:{BF,Z,AC,AZ,AT,AJ,BP}),

(W:{J,AI}),

(X:{AH,BG,L,T,AE,BY}),

(Y:{BF,X,AC,AV,AZ,AT,AJ,BL}),

(Z:{BF,X,Z,AZ,AT,AJ,BP})}

L= {(A,{pumabus}),

(AA,{alberca}),

(AB,{alberca}),

(AC,{facultad}),

(AD,{pumabus}),

(AE,{facultad}),

(AF,{museo}),

(AG,{pumabus}),

(AH,{facultad}),

(AI,{pumabus}),

(AJ,{facultad}),

(AK,{facultad}),

(AL,{pumabus}),

(AM,{pumabus}),

(AN,{facultad}),

(AO,{pumabus}),

(AP,{pumabus}),

(AQ,{alberca}),

(AR,{pumabici}),

(AS,{museo}),

(AT,{facultad}),

(AU,{pumabus}),

(AV,{facultad}),

(AW,{museo}),

(AX,{estadio}),

(AY,{facultad}),

```
(AZ,{facultad}),
(B,{museo}),
(BA,{pumabus}),
(BB,{estadio}),
(BC,{facultad}),
(BD,{pumabus}),
(BE,{alberca}),
(BF,{facultad}),
(BG,{facultad}),
(BH,{pumabus}),
(BI,{museo}),
(BJ,{estacion}),
(BK,{pumabus}),
(BL,{facultad}),
(BM,{estadio}),
(BN,{pumabus}),
(BO,{estadio}),
(BP,{facultad}),
(BQ,{estacion}),
(BR,{estadio}),
(BS,{alberca}),
(BT,{pumabus}),
(BU,{pumabus}),
(BV,{pumabus}),
(C,{pumabus}),
(CB,{pumabus}),
(CC,{estación}),
(CF,{pumabus}),
(CG,{metrobus}),
```

(CH,{estacion}),

(CI,{facultad}),

(CJ,{metrobus}),

(CK,{pumabus}),

(D,{museo}),

(E,{museo}),

(F,{pumabus}),

(G,{estadio}),

(H,{estacion}),

(I,{museo}),

(J,{estadio}),

(K,{museo}),

(L,{facultad}),

(M,{museo}),

(N,{pumabus}),

(O,{estadio}),

(P,{pumabus}),

(Q,{facultad}),

(R,{estadio}),

(S,{museo}),

(T,{facultad}),

(U,{pumabus}),

(V,{museo}),

(W,{museo}),

(X,{facultad}),

(Y,{pumabus}),

(Z,{facultad}),}

L={(A,{pumabus}),

(AA,{alberca}),

```
(AB,{alberca}),
(AC,{facultad}),
(AD,{pumabus}),
(AE,{facultad}),
(AF,{museo}),
(AG,{pumabus}),
(AH,{facultad}),
(AI,{pumabus}),
(AJ,{facultad}),
(AK,{facultad}),
(AL,{pumabus}),
(AM,{pumabus}),
(AN,{facultad}),
(AO,{pumabus}),
(AP,{pumabus}),
(AQ,{alberca}),
(AR,{pumabici}),
(AS,{museo}),
(AT,{facultad}),
(AU,{pumabus}),
(AV,{facultad}),
(AW,{museo}),
(AX,{estadio}),
(AY,{facultad}),
(AZ,{facultad}),
(B,{museo}),
(BA,{pumabus}),
(BB,{estadio}),
(BC,{facultad}),
```

(BD,{pumabus}),

(BE,{alberca}),

(BF,{facultad}),

(BG,{facultad}),

(BH,{pumabus}),

(BI,{museo}),

(BJ,{estacion}),

(BK,{pumabus}),

(BL,{facultad}),

(BM,{estadio}),

(BN,{pumabus}),

(BO,{estadio}),

(BP,{facultad}),

(BQ,{estacion}),

(BR,{estadio}),

(BS,{alberca}),

(BT,{pumabus}),

(BU,{pumabus}),

(BV,{pumabus}),

(C,{pumabus}),

(CB,{pumabus}),

(CC,{estación}),

(CF,{pumabus}),

(CG,{metrobus}),

(CH,{estacion}),

(CI,{facultad}),

(CJ,{metrobus}),

(CK,{pumabus}),

(D,{museo}),

(E,{museo}),

(F,{pumabus}),

(G,{estadio}),

(H,{estacion}),

(I,{museo}),

(J,{estadio}),

(K,{museo}),

(L,{facultad}),

(M,{museo}),

(N,{pumabus}),

(O,{estadio}),

(P,{pumabus}),

(Q,{facultad}),

(R,{estadio}),

(S,{museo}),

(T,{facultad}),

(U,{pumabus}),

(V,{museo}),

(W,{museo}),

(X,{facultad}),

(Y,{pumabus}),

(Z,{facultad})}

AP={A,AA,AB,AC,AD,AE,AF,AH,AI,AJ,AK,AL,AM,AN,AO,AP,AQ,AS,AT,AU,AV,AW,AZ,

B,BC,BD,BE,BF,BG,BI,BJ,BL,BP,BQ,BR,BT,BU,BV,BW,BX,BY,BZ,CA,CB,CC,CD,CE,

CF,CG,CH,CI,CJ,CK,CL,D,E,F,H,I,J,K,L,M,N,P,S,T,V,W,X,Y,Z,estacion,metrobus

pumabus,facultad,museo,estadio,alberca}

Apéndice E

Vista del prototipo funcional

La siguiente página muestra el prototipo con la vista del navegador de Google Maps:

1. Formulario para selección del centro del mapa dada una dirección
2. Vista del mapa
3. Formulario para selección de las palabras clave
4. Botones en el siguiente orden de izquierda a derecha:
   - Botón para búsqueda de lugares por palabra clave
   - Botón para generación de matriz de adyacencia por proximidad
   - Botón para borrar palabreas clave
   - Botón para generación de matriz de distancias entre puntos
   - Botón para visualización en mapa de la ruta desplegada
   - Botón para traducción de estructura y fórmula a NuSMV
   - Botón para traducción de estructura y fórmula a NuSMV
   - Botón para descargar código de NuSMV
   - Botón para generar lista de lugares a JSON
5. Formulario para selección de archivo respuesta de NuSMV
6. Vista de lugares de la ruta
7. Vista para despliegue de lugares de interés
8. Botones de selección de restricciones
9. Vistas de selección de restricciones
10. Botón para selección de orden en generación de fórmula
11. Botón para reseteo de restricciones

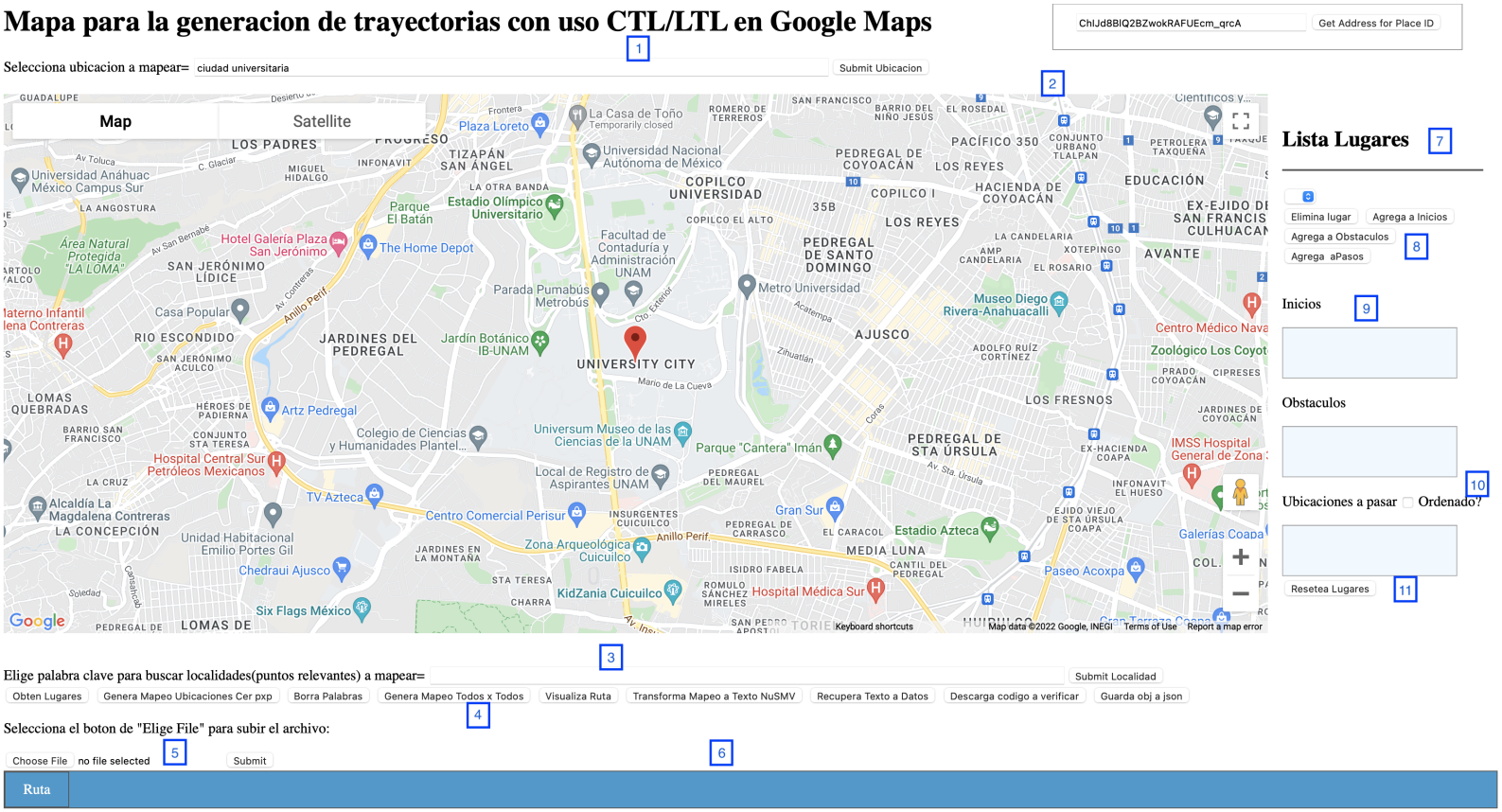


Figura E.1. Aplicación prototipo en el navegador

Bibliografía

[1] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchigliaa, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacche. **NuSMV 2: An OpenSource Tool for Symbolic Model Checking**. 2004. xi, 10, 11, 53

[2] Michael Rice and Vassilis Tsotras. **Exact graph search algorithms for generalized traveling salesman path problems**. page 344–355. Springer, 2012. 1, 11

[3] Heiko Schilling. **TomTom Navigation – How mathematics help getting through traffic faster**, 08 2012. 21st International Symposium on Mathematical Programming. 1

[4] E. M. Clarke and A. Emerson. **Synthesis of synchronization skeletons for branching time temporal logic**. page 52–71. Springer-Verlag, 1981. 1

[5] Dov Gabbay. **Saul A. Kripke. Semantical considerations for modal logics. Proceedings of a Colloquium on Modal and Many-valued Logics, Helsinki, 23-26 August, 1962, Acta Philosophica Fennica1963, pp. 83–94.** *Journal of Symbolic Logic*, **34**(3):501–501, 1969. 1, 20

[6] Matthias Felleisen. **On the expressive power of programming languages**. *Science of Computer Programming*, **17**(1):35–75, 1991. 3

[7] Rolf H. Mohring and Heiko Schilling. **Acceleration of Shortest Path and Constrained Shortest Path Computation**. TU Berlin Germany, 2005. 5

[8] Marcin Kamiński, Paul Medvedev, and Martin Milanič. **Shortest Paths between Shortest Paths and Independent Sets**. In Costas S. Iliopoulos and William F.

Smyth, editors, *Combinatorial Algorithms*, pages 56–67, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 5

[9] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks*, pages 19–80. Springer International Publishing, Cham, 2016. 5, 11

[10] P. Blackburn, J.F.A.K. van Benthem, and F. Wolter. *Handbook of Modal Logic*. ISSN. Elsevier Science, 2006. 7, 20, 22, 23, 65

[11] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, London,England, 2008. 7, 9, 37

[12] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchigliaa, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacch. **NuSMV: a new Symbolic Model Verifier**, 2008. 8, 10, 38

[13] Google. **Google Maps**. 8, 48, 65

[14] Martin Leucker. **On Model Checking Synchronised Hardware Circuits**. In He Jifeng and Masahiko Sato, editors, *Advances in Computing Science — ASIAN 2000*, pages 182–198, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. 9

[15] Edmund Clarke, Anubhav Gupta, Himanshu Jainxs, and Helmut Veith. **Model Checking: Back and Forth between Hardware and Software**. 2005. 9

[16] Yefei Shao. **Formal Model and Analysis of Sliding Window Protocol Based on NuSMV**. 2009. 9

[17] Venkat Margapuri Kai Zhao and Mitchell Neilsen. **Model Checking Mutual Inclusion and Mutual Exclusion Algorithms**. 2021. 9

[18] Meer Balach Jamali and Syed Zaffar Iqbal. **Testing Fuzzy Logic Street Light System using Uppaal**. *International Journal of Innovative Science and Research Technology*, 2018. 9

[19] Cabodi Gianpiero, Loiacono Carmelo, Palena Marco, Pasini Paolo, Patti Denis, Quer Stefano, Vendraminetto Danilo, Biere Armin, and Heljanko Keijo.

Hardware Model Checking Competition 2014: An Analysis and Comparison of Model Checkers and Benchmarks. *Journal on Satisfiability, Boolean Modeling and Computation*, 2014. 9

[20] Hans Henrik Løvengreen. *Introduction to SPIN*. 2006. Concurrent Systems ; Conference date: 01-01-2006. 10

[21] Sanjeev Arora and et al. *Computational complexity: A modern approach*. 2006. 10

[22] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. **Temporal logic motion planning for dynamic robots**. 2008. 11, 64, 65

[23] R. Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006. 15

[24] E. Clarke, Sumit Jha, Yuan Lu, and Helmut Veith. **Tree-like counterexamples in model checking**. pages 19– 29, 02 2002. 25

[25] Franco Mazzanti and Alessio Ferrari. **Ten Diverse Formal Models for a CBTC Automatic Train Supervision System**. *Electronic Proceedings in Theoretical Computer Science*, **268**:104–149, Mar 2018. 64

[26] Dominik Bucher, David Jonietz, and Raubal Martin. *A Heuristic for Multi-modal Route Planning*, pages 211–229. 10 2017. 64

[27] Dominik Bucher, Francesca Mangili, Francesca Cellina, Claudio Bonesana, David Jonietz, and Raubal Martin. **From location tracking to personalized eco-feedback: A framework for geographic information collection, processing and visualization to promote sustainable mobility behaviors**. *Travel Behaviour and Society*, **14**:43–56, 01 2019. 65

[28] Ren Wang, Mengchu Zhou, Kai Gao, Ahmed Alabdulwahab, and Muhyaddin Rawa. **Personalized Route Planning System Based on Driver Preference**. *Sensors*, **22**:11, 12 2021. 65

Glosario

API Interfaz de programación de aplicación. 13

CTL Lógica de calculo de árbol (Computation Tree Logic). 9

DPLL Algoritmo Davis–Putnam–Logemann–Loveland. 10

HWMCC The Hardware Model Checking Competition. 9

LTL Lógica de temporalidad lineal (Lineal Temporal Logic). 10

NuSMV New Symbolic Model Verifier. 8

OBDD Diagrama ordenado de decisión binaria(Ordered Binary Decision Diagrams). 10

PCTL Lógica de calculo de árbol probabilistica (Probability Computation Tree Logic). 9

PT Planificación de trayectorias. 1

RTCTL Real Time Computation Tree Logic. 10

SAT Problemas de Satisfabilidad Booleana. 10

VM Verificación de modelos. 1