UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"Representación de Planes de Evacuación usando Answer Set Programming"

T E S I S

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

José Miguel Leyva Cruz

DIRECTORES DE TESIS:

Dra. Claudia Zepeda Cortés

M.C. Wendy Yaneth García Martínez

HUAJUAPAN DE LEÓN, OAX. MARZO DE 2007

UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"Representación de Planes de Evacuación usando Answer Set Programming"

T E S I S

PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

José Miguel Leyva Cruz

Jurado

Dr. Nombre del Presidente

Presidente

M.C. Wendy Yaneth García Martínez

Vocal

Dr. Nombre del Secretario

Secretario

HUAJUAPAN DE LEÓN, OAX. MARZO DE 2007

A mis padres, Elsa y Javier por su amor y cariño. Por brindarme su confianza y apoyo, lo cual me alentó a seguir adelante a pesar de todos mis tropiezos. Gracias infinitas.

A mis hermanos, Noemí y Javier por estar siempre apoyándome.

A mis sobrinos con todo mi cariño, como prueba de que siempre hay que alcanzar las metas a pesar de los obstáculos que se pongan en el camino.

A mi abuelita Luz por estar siempre conmigo en las buenas y en las malas.

A la Dra. Claudia Zepeda Cortés, por la confianza puesta en mí para realizar juntos este objetivo, por su tiempo y apoyo.
A la M.C. Wendy Yaneth García Martínez por ayudarme a cumplir este objetivo, pero sobretodo y lo más importante, por su amistad y cariño durante este largo camino. Mil gracias!.
A los profesores M.C Verónica López Rodríguez, M.C María Auxilio Medina Nieto y M.C. Ricardo Rodríguez Ruíz, por su tiempo y colaboración para realizar este trabajo de tesis.

A todos mis amigos, con quienes inicié y cumplí esta meta, gracias por su amistad y cariño.

A todos aquellos también que conocí en este camino y que compartimos tantas experiencias.

A aquéllos que están por cumplir su meta les deseó la mejor de la suerte y no se den por vencidos.

A tantas y tantas personas que a lo largo de este camino me brindarón su amistad y cariño, gracias.

Índice general

Capítulo 1

Introducción

El razonamiento de sentido común es el tipo de razonamiento que todos nosotros desempeñamos diariamente en el mundo en base al conocimiento que tenemos acerca del mundo mismo y la habilidad para poder usar dicho conocimiento. Todo este conocimiento acerca de objetos, eventos, espacios, tiempo y estados puede usarse para hacer predicciones, explicar qué observamos y planear qué hacer [10].

El estudiar este razonamiento tiene dos razones: una razón científica y otra práctica. En particular, nuestro principal interés para el desarrollo de este trabajo se basa en la razón práctica, nos interesa basarnos en el razonamiento del sentido común para poder modelar el problema de encontrar planes de evacuación en una situación de riesgo utilizando las herramientas existentes de tal forma que pueda ser automatizado o procesado automáticamente por una computadora y que sirva de apoyo para la generación de planes y la toma de decisiones. Para nosotros un plan corresponde a la solución de un problema de planificación. En un problema de planificación estamos interesados en encontrar la secuencia de acciones que nos llevan de un estado inicial a un estado final o meta.

Para modelar y solucionar problemas de generación de planes de evacuación se deben considerar algunos aspectos como:

- **La existencia de diferentes escenarios.** Un escenario es un conjunto de características que describen el dominio de nuestro problema. Un mismo dominio puede tener diferentes escenarios. Por ejemplo teniendo como dominio un conjunto de poblaciones y un conjunto de caminos que conectan a las poblaciones entre si, un escenario sería que exista una ruta de evacuación de la población A a la población B cruzando por la población C. Otro posible escenario sería el que exista una ruta directa entre la población A y la población B.

- **Excepciones.** Una excepción es algún evento o situación inesperada que puede afectar las condiciones actuales de nuestro dominio. Por ejemplo, existen rutas de evacuación definidas por los gobiernos o por algunas organizaciones internacionales y que sirven para que los vehículos lleguen a los refugios por lo que una excepción sería que un sismo bloqueara alguna de estas rutas y que el autobús no pudiera pasar.

- **Restricciones.** Una restricción es una característica que limita algún aspecto de nuestro dominio. Por ejemplo, si la ruta de evacuación es de un solo carril se debe de limitar el paso de vehículos en una sola dirección, por lo que no pueden pasar dos vehículos que vayan en direcciones contrarias al mismo tiempo.

- **Preferencias.** Una preferencia es la elección entre varias alternativas de alguna que ayude a la solución del problema. Por ejemplo la capacidad del vehículo es limitada por lo que se tiene que dar preferencia a poblaciones con mayor riesgo para ser evacuadas primero.

- **Conocimiento incompleto.** Por ejemplo, existen caminos que pertenecen a una ruta de evacuación, pero en un determinado momento ocurre un evento que ocasiona que alguno de estos caminos se bloquee. Se tienen caminos que pueden servir

como rutas alternas, sólo que de éstos no se tiene alguna información o evidencia de su estado actual, sin embargo en tal caso se puede asumir que no están bloqueados y pueden ser utilizados como ruta de evacuación.

- **Costos.** Es el consumo total de medios necesarios para resolver el problema. Por ejemplo tiempo, gasolina, tipo de vehículo, cantidad de vehículos, etc.

En el presente trabajo utilizaremos *Answer Set Programming* (ASP) [7] para modelar problemas de planes de evacuación para situaciones de riesgo considerando algunos de los aspectos antes mencionados. ASP es un lenguaje declarativo útil para la representación del conocimiento, razonamiento y solución de problemas que tienen que ver con el sentido común. ASP además cuenta con un enfoque llamado *Answer Set Planning (AS Planning)* [8] que permite modelar y resolver problemas de planificación. Se utilizarán dos de los sistemas más conocidos para ASP: SMODELS y DLV con la finalidad de conocer la flexibilidad de dichos sistemas para modelar los problemas de planificación. Nos interesa utilizar ASP dado que actualmente no hay tantas aplicaciones que apoyen el uso de dicho lenguaje en la solución de problemas de generación de planes de evacuación. El poder modelar este tipo de problemas usando ASP permite ampliar el número de dominios donde ASP puede ser aplicado. Esto último resulta importante debido a que actualmente ASP cuenta con bastantes resultados teóricos pero el número de aplicaciones reales aún es reducido.

En [2] se muestra una primera evidencia de que es posible modelar el problema de generación de planes de evacuación usando ASP. De aquí que ASP puede servir para representar problemas con diferentes escenarios, en este caso situaciones de riesgo, y así obtener los planes de evacuación. En dicho trabajo se modelan con ASP algunos escenarios inspirados en la situacion de riesgo del volcán Popocatépetl para obtener los planes de evacuación. Sin embargo, estos escenarios sólo consideran unos cuantos ele-

mentos: poblaciones en riesgo, poblaciones donde se encuentran los refugios, autobuses que son usados para evacuar a la población, caminos en un solo sentido y que conectan a las poblaciones. Además tanto los caminos como los autobuses tienen una capacidad ilimitada. De manera general, en [2] se presentan fundamentos teóricos de que ASP puede ser utlizado para modelar problemas de generación de planes de evacuación para la situación de riesgo del volcán Popocatépetl. Sólo que los modelos de los ejemplos prácticos utilizados en dicho trabajo fueron muy pocos, además de que las condiciones de dichos modelos son ideales. Por lo tanto, la motivación del presente trabajo es extender el uso de ASP en la generación de planes de evacuación para situaciones de riesgo de manera general y no enfocándonos solamente al caso del volcán Popocatépetl. Además llevar a la práctica el modelado de escenarios que estén más apegados al mundo real con respecto a los ejemplos modelados en [2], por ejemplo tomar en cuenta la capacidad de los refugios, caminos y autobuses, número de habitantes en los lugares de riesgo, sentido de los caminos, etc. Una vez ampliados los escenarios se deben encontrar finalmente los planes de evacuación.

En conclusión, nuestros principales objetivos son: *modelar problemas de generación de planes de evacuación en situaciones de riesgo utilizando ASP mediante la implementación de escenarios más reales con respecto a [2], generalizándolos a cualquier situación en riesgo y no sólo para el caso del volcán Popocatépetl. Además, analizar qué tan flexibles son los sistemas SMODELS y DLV para llevar a la práctica el modelado de problemas de generación de planes de evacuación en ASP.*

En el capítulo 2 presentamos el marco teórico para entender el trabajo realizado en esta tesis; en el capítulo 3 se muestra la definición de algunos elementos que son necesarios en una situación de riesgo en base en información propuesta por organizaciones internacionales y basándonos en estos elementos definiremos el dominio para nuestro problema de modelado; en el capítulo 4 comenzaremos con la implementación práctica

de un escenario definiendo primero cada uno de los elementos de *AS Planning* para posteriormente implementarlos utilizando los sistemas SMODELS y DLV, para este último utilizaremos el enfoque de planificación llamado DLVK; en el capítulo 5 se implementará el dominio definido en el capítulo 3 utilizando SMODELS; en el capítulo 6 se implementará nuevamente el dominio definido en el capítulo 3 pero ahora utilizando DLV. Finalmente en el capítulo 7 se presentarán las conclusiones generales de este trabajo de tesis.

Capítulo 2

Marco Teórico

En este capítulo se presenta una síntesis de los antecedentes teóricos y prácticos de nuestro trabajo de tesis. Comenzaremos explicando qué es el razonamiento de sentido común, su definición y las razones que le dan importancia para ser estudiado. Presentaremos información referente al lenguaje de programación declarativo *Answer Set Programming* (ASP), además de las características y enfoques que tiene ASP para manejar los problemas de generación de planes de evacuación para situaciones de riesgo, principalmente el enfoque *Answer Set Planning (AS Planning)*.

2.1. Razonamiento de Sentido Común

El razonamiento de sentido común es el tipo de razonamiento que todos nosotros desempeñamos diariamente en el mundo. Por ejemplo, podemos predecir que si una persona entra a una cocina, entonces después de esto la persona está en la cocina, o bien, que si hay alguien que en un momento dado está sosteniendo un periódico y camina a otra posición, entonces el periódico se encontrará en la misma posición que la persona. También sabemos que una persona no puede estar en dos lugares al mismo tiempo. Para una persona inferir todo esto es una tarea muy fácil y los ejemplos anteriores dan la

impresión de que el razonamiento de sentido común es algo simple, sin embargo realizar la automatización de dicho razonamiento para ser implementado en una computadora es algo mucho más complejo. El razonamiento del mundo requiere un gran conocimiento acerca del mundo mismo y la habilidad para poder usar ese conocimiento. Por ejemplo, nosotros sabemos acerca de objetos, eventos, espacios, tiempo y estados mentales lo que nos permite utilizar este conocimiento para hacer predicciones, explicar qué observamos y planear qué hacer [10].

¿Por qué estudiar el razonamiento de sentido común? Hay una razón práctica y otra científica. En la práctica, la automatización del razonamiento de sentido común tiene un gran rango de aplicaciones como en las interfaces de usuario inteligentes, el procesamiento del lenguaje natural y la robótica. Por ejemplo, indicar a un robot que la mejor opción ante un obstáculo que se encuentra frente a él es dar un giro de 180 grados para que pueda librarlo. En lo científico, el razonamiento de sentido común es una capacidad importante de la inteligencia que soporta muchas otras capacidades de alto nivel. La habilidad de entender qué es lo que está sucediendo en una historia, envuelve crucialmente el razonamiento de sentido común [10]. Por ejemplo, si el lobo sopla y sopla sobre la casa de paja de uno de los tres cochinitos, se pueden saber las consecuencias que se tendrán.

Actualmente, investigaciones en el área de Inteligencia Artificial (IA) han utilizado algunos aspectos del comportamiento humano para ser representados en una computadora, lo cual es uno de los intereses principales de esta tesis. Esto es, nos interesa ayudar a entender y describir el razonamiento de sentido común de tal forma que se alcance un conocimiento más amplio de un dominio de estudio y este razonamiento pueda ser automatizado, o procesado automáticamente por una computadora utilizando ASP.

2.1.1. Puntos Claves del Razonamiento de Sentido Común

El razonamiento de sentido común parece ser un proceso que se realiza de manera natural, pero como ya se mencionó anteriormente, este proceso es realmente complejo. Para poder realizar la automatización del razonamiento de sentido común acerca de algún escenario, según [10], lo primero es construir una representación de dicho escenario y considerar cuatro entidades básicas. Primero, representar objetos del mundo y agentes tales como personas, vehículos, poblaciones, etc. Segundo, considerar propiedades del mundo que cambien con el tiempo, por ejemplo si se deja la llave abierta de un lavabo llegará el momento en que el agua alcance la orilla del lavabo y ésta se derrame. Tercero, representar eventos o acciones que ocurren en el mundo, como poder representar que una persona se encuentra caminando. Cuarto, representar el tiempo, por ejemplo representar la posición de una persona en un tiempo T y su ubicación en un tiempo T+1.

Además de las entidades básicas mencionadas anteriormente, también en [10] y en [1] se mencionan otros elementos que nosotros consideramos importantes para llevar a cabo el modelado. Uno de ellos es la concurrencia de los eventos, es decir, considerar que ciertos eventos no pueden ser realizados en un mismo momento, por ejemplo una persona no puede estar comiendo en París y durmiendo en México en el mismo instante de tiempo. Así mismo, también se deben de representar las precondiciones necesarias para que se pueda realizar un evento o acción, por ejemplo, si un vehículo va a comenzar la marcha, éste debe tener gasolina. Otro aspecto a considerar es la ley de la inercia de sentido común, esto es, un objeto o persona tiende a permanecer en el mismo estado a menos que sea afectado por algún evento o acción, por ejemplo un vehículo permanece estacionado hasta que no se aplique la acción arrancar que ocasione que el vehículo se ponga en marcha.

Una vez realizada una representación del escenario en base en las entidades propuestas

en [10], se puede automatizar el razonamiento del sentido común, esto es, por medio de un algoritmo o reglas formales tomar dicha representación como entrada y producir las salidas. Para este caso las salidas corresponden al conjunto de planes que ayudan a la solución del problema de evacuación.

2.2. Planificación utilizando *Answer Set Programming*

En un problema de planificación estamos interesados en encontrar la secuencia de acciones que nos lleven de un estado inicial a un estado final. Aprovechando las propiedades que ASP nos brinda para representar el sentido común, podemos modelar varias de las características que tiene un problema de planes de evacuación, donde los *answer sets* generados por el lenguaje corresponden al conjunto de planes que ayudan a la solución del problema. En [2] se muestra una primera evidencia de que es posible modelar el problema de generación de planes de evacuación usando ASP.

ASP es un lenguaje de programación lógica declarativo que permite representar el conocimiento. Fue definido en 1987 por Gelfond y Lifschitz [8], posee sólidos fundamentos teóricos basados en ideas de muchas áreas de la IA y la lógica matemática, se puede realizar la representación de restricciones y excepciones, permite expresar un problema de manera general, es decir, una familia de problemas.

Dos de los sistemas más populares para calcular los *answer sets* son: SMODELS[1] y DLV[2] el cual cuenta con DLVK[3] que es un *front-end* diseñado para modelar problemas de planificación.

SMODELS es un sistema para ASP, consiste en la implmentación de la semántica de modelos estables para programas lógicos y *lparse* el cual es un *front-end* que transforma

[1] http://www.tcs.hut.fi/Software/smodels
[2] http://www.dbai.tuwien.ac.at/proj/dlv
[3] http://www.dbai.tuwien.ac.at/proj/dlv/K/

los programas del usuario de tal manera que SMODELS los entienda. En [13] se puede encontrar más información sobre SMODELS.

DLV es un sistema basado en el lenguaje de programación declarativo *datalog* que es conocido por ser una herramienta conveniente para la representación del conocimiento. DLVK es un *front-end* implementado sobre la lógica de la programación del sistema DLV y el lenguaje de acción K el cual es un lenguaje de planeación basado en lógica. En [4] se presenta más información sobre estos sistemas.

Por otra parte se han propuesto diferentes enfoques en ASP para representación de problemas que tienen que ver con preferencias, actualización, argumentación, planificación, etc. Específicamente ASP cuenta con un enfoque llamado *AS Planning* que nos permite modelar problemas de planificación de manera natural y elegante.

2.2.1. *Answer Set Planning*

Como ya se mencionó anteriormente *AS Planning* es un enfoque que posee ASP el cual nos ayuda en la representación de problemas de planificación. En este caso utilizaremos este enfoque para modelar problemas de evacuación y generar los planes que puedan apoyar en la toma de decisiones. El llevar a cabo el modelado en *AS Planning* tiene básicamente tres etapas: en la primera se definen los elementos del dominio como son: fluentes, acciones, funciones de transición, el estado inicial y el estado final o meta; dichos elementos serán explicados en la siguiente subsección. Seguido de esto se modela utilizando alguno de los lenguajes de acción. En este caso utilizaremos el lenguaje de acción A como se mostrará más adelante. Finalmente, la tercera etapa consiste en llevar estos elementos a la implementación utilizando algunos de los sistemas como SMODELS y DLV. En la referencia [1] se puede encontrar más información sobre *AS Planning*.

Primera etapa del modelado utilizando *AS Planning*

La primera etapa de *AS Planning* se basa en la identificación de los siguientes elementos:

- Fluentes que representan propiedades del mundo. Un estado del mundo es un conjunto de fluentes.
- Acciones, eventos que cambian el estado del mundo.
- Función de transición que indica la transición de un estado a otro dada la ejecución de una acción. Dentro de la función de transición se definen las condiciones previas necesarias para que una acción pueda ser ejecutada y las consecuencias de ejecución que son aquellos efectos que tienen las acciones sobre los fluentes una vez que han sido ejecutadas.
- Estado inicial, donde se definen las condiciones iniciales de nuestro dominio.
- Estado final o meta, corresponde al objetivo que se quiere alcanzar.

Ejemplo 2.1. En la figura 2.1 se tienen los nodos 1, 2 y 3. El nodo 1 está conectado al nodo 2 y este último está conectado al nodo 3. El nodo 1 se encuentra dentro de la zona de riesgo y el nodo 3 es un refugio. En el nodo 1 se ubican dos autobuses para poder evacuar a las personas. Un fluente podría ser *posicion_bus* el cual representa la posición del autobús en un determinado instante. El estado corresponderá a indicar si el autobús se encuentra en la posición: *1, 2 o 3*. Una de las acciones podría ser *avanzar_bus* la cual define que el autobús está avanzando de una posición inicial a una posición final. La función de transición indica que el autobús cambia de la posición A a la posición B cuando la acción *avanzar_bus* se ejecuta. Esta acción es posible si la posición A es diferente a la posición B y además si el autobús se encuentra inicialmente en la posición

A. El estado inicial corresponde a definir la posición inicial del autobús para este caso *posición 1*. El estado final o meta sería que el autobús llegue a la *posición 3*.

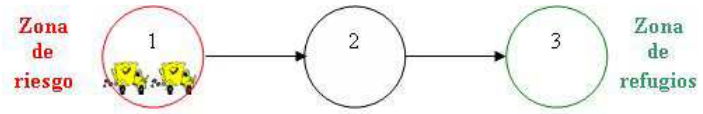

Figura 2.1: Red de nodos

De aquí que *AS Planning* puede servir para representar problemas con diferentes escenarios, en este caso situaciones de riesgo, y así obtener los planes de evacuación. En [2] se muestra una primera evidencia de que es posible modelar problemas de generación de planes de evacuación usando *AS Planning*.

Segunda etapa del modelado utilizando *AS Planning*

En la segunda etapa del enfoque *AS Planning* se modela utilizando lenguajes de acción. Los lenguajes de acción son utilizados para especificar los efectos de las acciones y en particular son usados para modelar problemas de planificación. Algunos de los lenguajes de acción son: A, B, C, K [8].

En este caso utilizaremos el lenguaje de acción A porque es suficiente para modelar los elementos definidos en el ejemplo 2.1. Para mayor información sobre la codificación utilizando alguno de los otros lenguajes de acción revisar las referencias [1, 8].

El alfabeto del lenguaje de acción A [8] consiste en dos conjuntos no vacíos: F y A, donde F es el conjunto de fluentes y A expresa el conjunto de acciones. Un fluente expresa propiedades del mundo y forma parte de la descripción de los estados del mundo. Por ejemplo, para indicar el estado de un autobús en una situación de riesgo se pueden tener los siguientes fluentes $\{autobúsASalvo,\ autobúsEvacuando\}$ y para el caso de una

población se pueden tener los siguientes fluentes {*poblaciónEnRiesgo, poblaciónASalvo*}, entonces el conjunto F quedaría de la siguiente manera:

F={*poblaciónEnRiesgo, poblaciónASalvo, autobúsASalvo, autobúsEvacuando*}

Un estado σ corresponde a un conjunto de fluentes. Si f es un fluente decimos que $\neg f$ se cumple en el estado σ si $f \notin \sigma$. Por ejemplo si σ={*poblaciónASalvo, autobúsASalvo*} y f=*poblaciónEnRiesgo*, entonces $\neg f$ se cumple ya que no se encuentra definido dentro de σ. Del mismo modo, si f es un fluente decimos que f se cumple en el estado σ si $f \in \sigma$. Por ejemplo si σ={*poblaciónASalvo, autobúsASalvo*} y f=*poblaciónASalvo* entonces decimos que f se cumple ya que se encuentra definido en σ.

El conjunto de acciones A puede ser el siguiente: A={*viajar, subir, manejar, cargar*}.

El lenguaje de acción A está constituido por tres sub-lenguajes: lenguaje de descripción del dominio, lenguaje de observaciones y lenguaje de consulta. Dado un problema de planificación, éste puede ser expresado por medio de una tripleta (D,O,G), donde D es el conjunto de descripción del dominio, O es el conjunto de observaciones y G es el conjunto de condiciones de meta [2, 1].

El conjunto de descripción del dominio usa proposiciones de efecto, las cuales tienen la siguiente forma:

$$\text{a } \textit{causes} \text{ f } \textit{if } p_1,...,p_n,\neg q_1,...,\neg q_m$$

donde:

- a es una acción.
- f es un fluente.
- p_1,...,p_n existen los fluentes en el estado.
- $\neg q_1$,...,$\neg q_m$ no existen los fluentes en el estado.

También se puede tener:

$$\text{a } \textit{causes } f$$

si no se necesitan condiciones para ejecutar la acción.

O bien:

$$\text{a } \textit{causes } f_1,f_2,...,f_n$$

si es que la acción ocasiona múltiples cambios en los fluentes. Las proposiciones de efecto definen una función de transición. El conjunto de descripción de dominio se denota por D.

El conjunto de observaciones tiene un conjunto de proposiciones de valor de la forma:

$$\text{f } \textit{after } a_1,...,a_n$$

El conjunto de observaciones se denota por O. Dado D y O podemos encontrar el conjunto de estados iniciales. Un estado inicial lo podemos denotar como σ_0. Entonces se puede tener la siguiente forma de representación de dichos estados iniciales:

$$\textit{initially } f$$

$$\textit{initially } f_1,...f_k$$

Un conjunto de observaciones O se dice que es *estado completo*, si O sólo consiste de proposiciones de la forma *initially* f.

El conjunto de condiciones de meta se denota por G, donde G contiene el conjunto de estados finales o meta. La representación de dichos estados finales es de la forma:

$$\textit{finally } \text{f}$$

$$\textit{finally } f_1,...f_k$$

Retomando el ejemplo 2.1. y utilizando el lenguaje de acción A, el modelado es el siguiente:

1. **Fluentes.**

   Para indicar la posición actual de un autobús se puede definir el siguiente fluente:

   posicion_bus(Bus, X)

2. **Acciones.**

   Para representar que un autobús avanza de la posición A a la posición B se puede definir la siguiente acción:

   avanzar_bus(Bus,A,B)

3. **Condiciones y proposiciones de efecto.**

   También llamadas consecuencias y condiciones de ejecución de las acciones.

   La acción *avanzar_bus* causa que la posición del autobús cambie de la posición A a la posición B si la posición actual del autobús es A. Además A y B son diferentes.

   avanzar_bus(Bus,A,B) ***causes*** *posicion(Bus,B)* ***if*** *posicion(Bus,A), A≠B*

4. **Condiciones iniciales.**

   La posición inicial de los autobuses queda representada de la siguiente forma:

   initially *posicion(bus1,1)*

   initially *posicion(bus2,1)*

5. **Estados finales o metas.**

   La posición final de los autobuses queda representada de la siguiente forma:

   finally *posicion(bus1,3)*

   finally *posicion(bus2,3)*

Los fluentes, acciones, condiciones y proposiciones de efecto pueden ser tantos como el problema que está siendo modelado lo requiera.

Tercera etapa del modelado utlizando *AS Planning*

Siguiendo con las etapas del modelado utilizando *AS Planing* finalmente se llega a la implementación. Para esto se pueden usar los sistemas SMODELS o DLV. El programa utlizando SMODELS y basándonos en el ejemplo 2.1. queda de la siguiente manera:

%Estados iniciales.

initially*(posicion(bus1,1)).*

initially*(posicion(bus2,1)).*

%Estados finales o meta.

finally*(posicion(bus1,2)).*

finally*(posicion(bus2,3)).*

%Fluentes.

fluent *(posicion(Bus,N)) :- bus(Bus), nodo(N).*

%Acciones.

action*(**act**(Bus,avanzar_bus(A,B))) :- camino(A,B), bus(Bus).*

%Condiciones y consecuencias de ejecución de las acciones.

%La acción avanzar_bus causa que la nueva posición del autobús Bus es B.

%La acción avanzar_bus no es posible si el autobús Bus no se encuentra en A.

caused*(posicion(Bus,B),**act**(Bus,avanzar_bus(A,B))):- bus(Bus), camino(A,B).*

caused*(**neg**(posicion(Bus,A)),**act**(Bus,avanzar_bus(A,B))) :- bus(Bus), camino(A,B).*

noaction_if*(**act**(Bus,avanzar_bus(A,B)),**neg**(posicion(Bus,A))):- bus(Bus),camino(A,B).*

Debido a que el ejemplo es muy pequeño se tiene como salida un solo *answer set* el cual corresponde a un plan. En el tiempo 1 el autobús a1 avanza del nodo 1 al nodo 2 lo mismo que el autobús *a2*. En el tiempo 2 el autobús *a1* avanza del nodo 2 al nodo 3, al igual que el autobús *a2*. En este tiempo ambos autobuses cumplen con la meta, la cual es llegar al nodo refugio 3 y es aquí donde se termina la ejecución del programa. En la tabla 2.1 se puede apreciar la secuencia de las acciones.

TIEMPO	ACCIONES	
1	(a1, avanzar_bus(1, 2))	(a2, avanzar_bus(1, 2))
2	(a1, avanzar_bus(2, 3))	(a2, avanzar_bus(2, 3))

Tabla 2.1: *Answer set* generado por SMODELS

El programa anterior sólo es una muestra de cómo se modela utilizando SMODELS. En los capítulos siguientes se mostrará un ejemplo más grande y se darán más detalles de la implementación.

2.3. Conclusiones

En este capítulo presentamos una síntesis de las bases teóricas de nuestra investigación de tesis con el objetivo de que el lector conozca las características y los enfoques que tiene ASP para modelar problemas de planificación, el cual será utilizado para generar planes de evacuación. En los siguientes capítulos presentaremos los resultados de llevar a la práctica estas bases teóricas.

Capítulo 3

Información para problemas de planes de evacuación

En este capítulo se presenta información referente a los elementos que se deben considerar en caso de una situación de riesgo cbasándonos en datos propuestos por organizaciones como la UNESCO y otros investigadores [5]. Posteriormente analizaremos dichos elementos para tomar aquéllos que consideramos más adecuados para realizar el modelado en ASP. Esto con la finalidad de delimitar el dominio y definir los escenarios sobre los cuales vamos a trabajar para finalmente generar los planes.

3.1. Elementos para definir un plan de evacuación

La UNESCO en 1985 afirmó que: "Los países deberían de prepararse para situaciones de desastre y proveer protección de ellos..."[11]. De aquí que la UNESCO recomienda a los gobiernos de cada país diseñar sus propios planes de evacuación. Dicha organización publicó una lista de elementos que se deben considerar en caso de una situación de riesgo y que pueden ayudar a los gobiernos de los países. Además también consideramos los elementos propuestos por Tjandra en [5].
Algunos de los elementos propuestos en [11] y [5] son:

- Identificar el tipo de sistema sobre el cual se va a realizar el plan de evacuación, por ejemplo si se trata de un aeropuerto, edificio, escuela, etc.
- Definir aquellas zonas o lugares de riesgo.
- Considerar el número de personas en las zonas de riesgo.
- Determinar los refugios a los cuales pueden ser evacuadas las personas en caso de una situación de riesgo.
- Identificar los caminos que pueden servir de rutas de evacuación, su estado, capacidad y condiciones.
- Agrupar a las personas, ya sea por sexo o edad.
- Definir puntos de intersección que pueden servir para que las personas esperen mientras son evacuadas.
- Determinar las fuentes de peligro y características del peligro.
- Considerar los medios de transporte o móviles disponibles y formas de controlar el tráfico.
- Clasificar el equipo y personal de servicio de emergencia.
- Contar con hospitales y servicios médicos para atender a las personas.
- Utilizar procesos de alerta.
- Determinar formas de comunicación para alertar en caso de peligro.

Identificar los elementos necesarios para definir los planes de evacuación nos permite conocer el dominio de nuestro problema e ir delimitándolo de acuerdo a la situación

de riesgo que se quiera analizar. Además se debe considerar que los elementos sean adecuados para ser modelados utilizando ASP.

3.2. Definición de escenarios

Los elementos del dominio para una situación de riesgo pueden llegar a ser muy extensos, como se puede ver en la sección anterior. Basándonos en estos elementos y con la finalidad de extender el trabajo realizado en [2] se propone delimitar el dominio para los problemas de los planes de evacuación tomando en cuenta los siguientes elementos:

- Capacidad en las poblaciones para recibir un número limitado de vehículos en un mismo instante de tiempo, principalmente en aquéllas que son consideradas como refugios.
- Número de habitantes en las poblaciones ubicadas dentro de la zona de riesgo.
- Vehículos con capacidad limitada para evacuar a las personas que se encuentran en la zona de riesgo.
- Los vehículos tendrán que ir y regresar de la zona de riesgo a los refugios, hasta que el número de habitantes sea igual a cero en las poblaciones en riesgo.
- Inicialmente en cada población ubicada dentro de la zona de riesgo se cuenta con un vehículo para poder evacuar a las personas.
- Tomar en cuenta la capacidad y el sentido de los caminos que sirven como ruta de evacuación.

Todo lo anterior con el objetivo de modelar escenarios con elementos más apegados a la realidad con respecto a los modelos implementados en [2].

3.3. Conclusiones

En este capítulo presentamos información acerca de algunos elementos que se deben de considerar en una situación de riesgo los cuales son propuestos por organizaciones internacionales tales como la UNESCO, la cual realiza esta lista basándose en experiencias surgidas en países alrededor del mundo y que pueden ayudar a otros países en caso de que sea necesario. Además de la información dada por la UNESCO consideramos la propuesta por Tjandra en [5] quien propone también algunos elementos para problemas de evacuación.

Finalmente se hizo una delimitación de los elementos del dominio seleccionando aquéllos que pueden ser modelados utilizando ASP con el objetivo de trabajar con escenarios más reales que en [2].

Capítulo 4

Modelando problemas de planes de evacuación

En este capítulo comenzaremos con la implementación práctica de un escenario con el objetivo de dar una idea de cómo se modela utilizando ASP. Primero se presenta el modelo utlizando *AS Planning* con la finalidad de definir cada uno de los elementos de dicho enfoque: condiciones iniciales, fluentes, acciones, la meta u objetivo, precondiciones y consecuencias de ejecución de las acciones . Se implementará dicho modelo utilizando dos de los sistemas más comunes para ASP: SMODELS y DLV, en este último utilizaremos el enfoque para planificación llamado DLVK. Para cada implementación se obtendrán los planes de evacuación que corresponden a los *answer sets*. En el capítulo siguiente se modelarán los elementos del dominio definidos en la sección 3.2 de este trabajo de tesis con el objetivo de ampliar lo propuesto en [2].

4.1. Modelado del problema de planificación en *AS Planning*

En la sección 3.2. del capítulo anterior identificamos los elementos para una situación de riesgo, seleccionando algunos de esos elementos para modelar los escenarios. En este

trabajo de tesis el dominio será la situación de riesgo en caso de erupción de un volcán. Cabe mencionar que la implementación propuesta en esta tesis puede ser utilizada para cualquier situación de riesgo, basta solamente con adecuar los programas a las características de cada dominio.

En el dominio que presentaremos en esta tesis un escenario corresponde a un conjunto de poblaciones unidas por caminos y que pueden servir como rutas de evacuación. Aquellas poblaciones que se encuentran cercanas al volcán serán consideradas como poblaciones en riesgo. También existen poblaciones que se consideran como refugios, las cuales se encuentran alejadas de la zona de riesgo. La red de poblaciones y caminos es representada por medio de un grafo no dirigido donde los nodos corresponden a las poblaciones (algunas de ellos a las poblaciones en riesgo y otros a los refugios) y los arcos a los caminos entre dichas poblaciones.

Ejemplo 4.1. Para ejemplificar el modelado en *AS Planning* se presenta la Figura 4.1 la cual consta de 3 poblaciones. La población 1 se encuentra dentro de la zona de riesgo y la población 3 es un refugio. Existe un solo autobús en la población 1 el cual tiene como meta u objetivo llegar al refugio. Los caminos conectan a la población 1 con la población 2 y ésta a su vez está conectada con la población 3. Los caminos pueden ser utilizados en ambas direcciones, tanto para ir hacia el refugio como para regresar a la zona de riesgo.

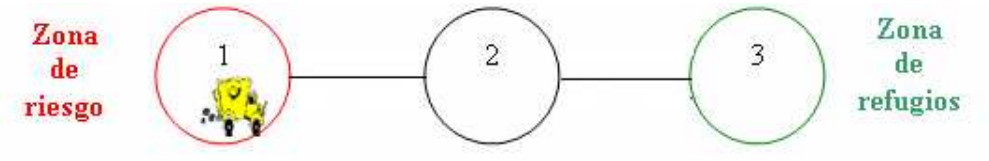

Figura 4.1: Red de poblaciones y caminos

Lo primero que se modela son las características de nuestro escenario. Esto se conoce

como conocimiento previo o *background knowledge.*

- ***Background knowledge.*** En el *background knowledge* se definen todas las poblaciones, indicando cuales son las poblaciones refugio, las poblaciones en riesgo, el número de autobuses que existen para la evacuación, los caminos que conectan a cada una de las poblaciones y el sentido de los caminos. El escenario de la figura 4.1 queda modelado de la siguiente manera:

  %Se definen los nodos que representan a las poblaciones.

  ```
  node(1).  node(2).  node(3).
  ```

  %Se declaran cuáles son las poblaciones en riesgo.

  ```
  risk_node(1).
  ```

  %Se define la población que es considerada como refugio.

  ```
  refuge_node(3).
  ```

  %Se declaran los caminos entre las poblaciones.

  %Conjunto de segmentos que permiten evacuar hacia la zona de refugios.

  %going_way(población inicial, población final).

  ```
  going_way(1,2).  going_way(2,3).
  ```

  %Conjunto de segmentos que permiten regresar a la zona de riesgo.

  %return_way(población inicial, población final).

  ```
  return_way(2,1).  return_way(3,2).
  ```

  %Se definen los autobuses.

  %bus(identificador del autobús).

  ```
  bus(a1).
  ```

Notesé que los caminos son en ambos sentidos así que se establecen segmentos tanto de ida como de regreso.

Una vez representado el escenario, se modelan cada uno los elementos necesarios que

definen a todo problema de planificación en *AS Planning*: fluentes, acciones, condiciones de ejecución de las acciones, consecuencias de ejecución de las acciones, condiciones iniciales y metas.

- **Fluentes.** El fluente considerado para este ejemplo es el siguiente:

  ```
  position(B,N)
  ```

  Este fluente indica la posición del autobús, donde B es el autobús y N la población en donde se ubica en un determinado momento. El número de fluentes puede ser tan grande como el problema lo requiera.

- **Estado inicial.** La posición inicial del autobús `a1` es la población 1.

  ```
  position(a1,1).
  ```

- **Estado final o meta.** En este caso la meta es que el autobús `a1` llegue al refugio (población 3).

  ```
  position(a1,3).
  ```

- **Acciones.** La acción que consideramos en este ejemplo es la siguiente:

  ```
  evacuate(B,I,F)
  ```

  indica que el autobús B se encuentra viajando de la población inicial I a la población final F.

  Esta acción no es necesariamente la única, el número de acciones puede ser tan grande como el problema de modelado lo requiera.

- **Condiciones de ejecución de las acciones.** Las condiciones para la acción definida anteriormente son las siguientes:

```
evacuate(B,I,F) if position(B,I), going_way(I,F).
```

El autobús B podrá evacuar de la población inicial I a la población final F si B se encuentra en la población incial I y además debe existir un segmento de camino que une a I con F.

- **Consecuencias de ejecución de las acciones.**

  Una vez realizada la acción `evacuate` se tienen dos consecuencias. La primera consecuencia es que la nueva posición del autobús B es la población final F.

```
position(B,F) after evacuate(B,I,F).
```

  La segunda consecuencia es que la posición de B ya no es la población inicial I. La negación de que la posición actual de B ya no es la población I se modela anteponiendo el signo - al fluente.

```
-position(B,I) after evacuate(B,I,F).
```

Como se mencionó anteriormente, tanto los fluentes como las acciones modeladas para la Figura 4.1 no son los únicos.

Como se puede apreciar en el ejemplo anterior el modelado en *AS Planning* es muy sencillo y las modificaciones necesarias simplemente consisten en agregar una o más líneas. Esta es una de las grandes ventajas que presenta ASP a diferencia de otros enfoques o algoritmos en los cuales el agregar o eliminar alguna característica requiere volver a construir todo el algoritmo. Además, esta ventaja permite que el modelo hecho para una situación de riesgo pueda ser utilizado en cualquier otra.

4.2. Modelado en SMODELS

En esta sección se presenta la implementación del escenario propuesto en el ejemplo 4.1. utilizando SMODELS. El código en SMODELS está divido básicamente en dos partes: una que es dependiente del dominio en la cual se definen las características del escenario junto con los fluentes, acciones, condiciones de ejecución, consecuencias de ejecución, estado inicial y meta. La otra parte es independiente del dominio. En esta última se definen las restricciones de ejecución y es necesaria para modelar cualquier problema de planificación utilizando SMODELS.

Ahora bien, recordemos que para nuestro ejemplo la posición inicial del autobús es la población 1 y la meta es que debe llegar a la población 3, la cual es un refugio. La implementación del escenario queda de la siguiente manera:

```
%José Miguel Leyva Cruz, 16/11/2006 UTM
%ejemplo1.sm

%Se declara una constante númerica n la cual representa el tiempo
%máximo posible que se requiere para alcanzar la meta. En cada
%instante de tiempo se ejecuta una acción.
const n=4.
time(1..n).

% BACKGROUND KNOWLEDGE
% Conjunto de poblaciones. Donde cada población es un nodo.
node(1).  node(2).  node(3).
```

```
%Lista de nodos considerados como refugios.
refuge_node(3).

%Conjunto de caminos.
%Conjunto de segmentos que permiten evacuar hacia la zona de refugios.
%going_way(población inicio, población final).
going_way(1,2).  going_way(2,3).

%Conjunto de segmentos que permiten regresar a la zona de riesgo.
%return_way(población inicio, población final)
return_way(2,1).  return_way(3,2).

%Se definen los autobuses.
%bus(identificador del autobús).
bus(a1).

%Condiciones iniciales. La posición inicial del autobús a1 es la
% población 1.
initially(position(a1,1)).

%Posición final o meta. La meta es que el autobús a1 llegue a la
%población 3.
finally(position(a1,3)).

%Se verifica que la meta sea alcanzada.
%No se quiere que no haya un plan, es decir, se quiere que haya un plan.
```

```
:- not plan.

%Existe un plan si se llega a la meta a lo más en n pasos.
plan :- goal(n).

%La meta se cumple en el tiempo T si no hay evidencia de que la
%meta no haya sido cumplida.
goal(T) :- time(T),
       not not_goal(T).

%La meta no se cumple en el tiempo T si no hay evidencia de que se han
%cumplido las condiciones finales (finally).
not_goal(T) :- time(T),
       literal(X),
       finally(X),
       not hold(X,T).

%%%FLUENTES
%Indica que la posición del autobús B1 es el nodo N.
fluent(position(B1,N)) :- bus(B1), node(N).

%%%ACCIONES
%evacuate: el autobús B1 viaja de la población inicial I a la
%población final F siempre que haya un camino de ida de I a F.
action(act(B1,evacuate(I,F))) :- going_way(I,F), bus(B1).
```

```
%%%CONSECUENCIAS DE EJECUCIÓN DE LAS ACCIONES
%La primera consecuencia de la acción evacuate es que la nueva posición
%del autobús B1 es la población final F.
caused(position(B1,F),act(B1,evacuate(I,F))):-
                                              bus(B1),
                                              going_way(I,F).
%La segunda consecuencia de la acción evacuate es que la posición
%del autobús B1 ya no es la población inicial I.
caused(neg(position(B1,I)),act(B1,evacuate(I,F))):-
                                              bus(B1),
                                              going_way(I,F).

%%%CONDICIONES DE EJECUCIÓN DE LAS ACCIONES
%La acción evacuate no es posible si el autobús B1 no está en la
%población inicial I y además debe existir un camino que una a la
%población I con la población F.
noaction_if(act(B1,evacuate(I,F)),neg(position(B1,I))):-
                                                   bus(B1),
                                                   going_way(I,F).


%%%RESTRICCIONES DE EJECUCIÓN
%La acción A no es ejecutable en el tiempo T si las condiciones de
%ejecución para A no se cumplen.
not_executable(A,T) :-
            action(A),
```

```
            time(T),
            noaction_if(A,F),
            hold(F,T).

%La acción A es ejecutable en el tiempo T si no hay evidencia
%de que A no pueda ser ejecutada.
executable(A,T) :-
       action (A),
       time (T),
       not not_executable(A,T).

% Es posible ejecutar la acción A en el tiempo T si no hay evidencia
%de que la meta no haya sido cumplida.
possible(A,T) :-
       action(A),
       executable(A,T),
       time(T),
       not goal(T).

%El valor del fluente F cambia su valor en el tiempo T+1 si la
%acción A afecta a F. Además la acción A es ejecutable y está
%ocurriendo en el tiempo T.
hold(F,T+1) :- literal(F),
       time(T),
       T<n,
       action(A),
```

```
      executable(A,T),
      occurs(A,T),
      caused(F,A).

%El fluente F siempre se cumple en el tiempo 1 si se define
%como condición inicial. Esto se cumple para todos los fluentes
%definidos como initially.
hold(F,1) :- literal(F), initially(F).

%El fluente F no se cumple en el tiempo 1 si no está definido
%como condición inicial. Esto se cumple para todos los fluentes
%que no están definidos como initially.
hold(neg(F), 1) :- fluent(F), not hold(F,1).

%Si F es un fluente, su valor contrario es -F.
%Recordemos que la negación de un fluente se escribe con el símbolo -.
contrary(F, neg(F)) :- fluent(F).

%Si -F es un fluente, su valor contrario es F.
contrary(neg(F), F) :- fluent(F).

%La literal G es un fluente.
literal(G) :- fluent(G).

%La negación de la literal G también es un fluente.
literal(neg(G)) :- fluent(G).
```

```
%La literal F mantiene su valor en el tiempo T+1 si no hay
%evidencia de que cambie a su valor contrario G. Esto se conoce como
%la ley de la inercia, un fluente mantiene su valor hasta que no haya
%evidencia de que éste haya cambiado.
hold(F, T+1) :-literal(F),
       literal(G),
       contrary(F,G),
       time(T),
       T<n,
       hold(F,T),
       not hold(G, T+1).

%La acción A ocurre en el tiempo T si no hay evidencia de que la acción
%no pueda ocurrir en dicho instante.
occurs(A,T) :-
       action(A),
       time(T),
       possible(A,T),
       not not_occurs(A,T).

%La acción A ocurre en el tiempo T si se verifica que A es posible y no
%existe evidencia de que no pueda ocurrir en dicho tiempo.
occurs(A,T) :-
       action(A),
       time(T),
```

```
       possible(A,T),
       verify(A,T),
       not not_occurs(A,T).

%Las reglas siguientes serán explicadas más detalladamente
%en el capítulo siguiente. Se colocan en este momento ya que son
%necesarias para generar los planes en SMODELS
sub_action(evacuate(I,F)) :- going_way(I,F).
not_occurs(act(B1,A1),T) :-
       action(act(B1,A1)), sub_action(A1),bus(B1),
       action(act(B1,A2)), sub_action(A2),
       time(T),
       occurs(act(B1,A2),T),
       neq(A1,A2).

%%%FIN DE LAS RESTRICCIONES DE EJECUCIÓN.
```

Para poder ejecutar el programa anterior se escribe la siguiente instrucción en la línea de comandos.

```
lparse < ejemplo1.sm | smodels 0
```

El parámetro `0` al final de la instrucción indica que queremos visualizar todos los posibles *answer sets* que se puedan generar de acuerdo a cada escenario.
Para nuestro ejemplo sólo se genera un único plan, el cual corresponde a un *answer set*. El autobús sale de la población en riesgo (nodo 1) y llega a la población refugio (nodo 3). Esto lo podemos ver en la tabla 4.1.

TIEMPO	ACCIONES
1	evacuate(a1, 1, 2)
2	evacuate(a1, 2, 3)

Tabla 4.1: *Answer set* generado por SMODELS

4.3. Modelado en DLVK

En esta sección modelaremos nuevamente el escenario del ejemplo 4.1 pero ahora utilizando el front-end de DLV llamado DLVK.

DLVK por ser un enfoque dirigido a modelar problemas de planificación permite que la implementación práctica de un problema sea más directo con respecto a SMODELS.

Para definir el *background knowledge* en DLVK se crea un archivo con extensión .bk en el cual se definen: el número de poblaciones, las poblaciones en riesgo, las poblaciones refugio, el número de autobuses, los caminos entre poblaciones y su sentido.

```
%José Miguel Leyva Cruz UTM 28/09/2006
%ejemplo1.bk
%BACKGROUND KNOWLEDGE
%Conjunto de poblaciones. Donde cada población es un nodo.
node(1).  node(2).  node(3).

%Lista de nodos considerados como refugios.
refuge_node(3).

%Lista de nodos en la zona de riesgo.
risk_node(1).
```

```
%Conjunto de segmentos.
%Conjunto de segmentos que permiten evacuar hacia la zona de refugios.
%going_way(población inicial, población final).
going_way(1,2).  going_way(2,3).

%Conjunto de segmentos que permiten regresar a la zona de riesgo.
%return_way(población inicial, población final).
return_way(2,1).  return_way(3,2).

%Se definen los autobuses.
%bus(identificador del autobús).
bus(a1).
```

Como se puede apreciar en el código anterior el modelado del *background knowledge* en DLVK es igual que en SMODELS. Pero una de las diferencias que presenta DLVK con respecto a SMODELS es que el modelado de los fluentes, las acciones, las consecuencias de ejecución de las acciones, las condiciones de ejecución de las acciones, el estado inicial y la meta se realiza por bloques, iniciando cada bloque con una palabra reservada que ya definida en DLVK y que corresponde a cada elemento mencionado anteriormente. Esto permite que el modelado sea más sencillo con respecto a SMODELS. Toda esta definición se realiza en un archivo con extensión .plan como veremos a continuación.

```
%José Miguel Leyva Cruz UTM 28/09/2006
%ejemplo1.plan
```

```
%En este bloque se definen todos los fluentes.
fluents:
    %%Indica que la posición del autobús B1 es el nodo N.
    position(B1,N) requires bus(B1), node(N).

%En este bloque se definen todas las acciones.
actions:
    %evacuate: el autobús B1 viaja de la población inicial I a la
    %población final F, siempre que haya un camino de ida de I a F.
    evacuate(B1, I, F) requires bus(B1), going_way(I,F).

%En este bloque se definen las condiciones y las consecuencias de
%ejecución para cada acción.
always:
%Condiciones de ejecución de las acciones.
    %La acción evacuate es posible si el autobús B1 está en la
    %población inicial I y además debe existir un camino que una a
    %la población I con la población F.
    executable evacuate(B1, I, F) if position(B1,I), going_way(I,F).
%Consecuencias de ejecución de las acciones.
    %La nueva posición del autobús B1 es la población final F.
    caused position(B1,F) after evacuate(B1,I,F).
    %La posición del autobús B1 ya no es la población inicial I.
    caused -position(B1,I) after evacuate(B1,I,F).

%Se define la inercia en los fluentes. Esto es que los fluentes permanecen
```

```
%con sus valores hasta que no sean afectados directamente por la
%ejecución de una acción.
    inertial position(B1,N).

%Se define la no concurrencia entre acciones. Esta macro será explicada%más
detalladamente en los siguientes capítulos.
    noConcurrency.

%En este bloque se definen las condiciones iniciales.
initially:
    %La posición inicial del autobús a1 es la población 1.
    position(a1,1).

%En este bloque se definen las condiciones finales o meta.
goal:
    %La meta es que el autobús a1 llegue a la población 3.
    %El parámetro que sigue después del signo de interrogación
    %indica el tiempo máximo posible que se requiere para alcanzar
    %la meta. En cada instante de tiempo se ejecuta una acción.
    position(a1,3) ? (2)
```

Para poder ejecutar el programa anterior se escribe la siguiente instrucción desde la línea de comandos.

```
DLV ejemplo1.plan ejemplo1.bk -FP
```

Para nuestro ejemplo sólo se genera un único plan, el cual corresponde a un *answer*

set. El autobús `a1` sale de la población en riesgo (nodo 1) y llega a la población refugio (nodo 3). Esto lo podemos ver en la tabla 4.2.

TIEMPO	ACCIONES
1	evacuate(a1,1,2);
2	evacuate(a1,2,3);

Tabla 4.2: *Answer set* generado por DLVK

4.4. Conclusiones

En este capítulo trabajamos con los sistemas más utilizados para ASP: SMODELS y DLV para implementar problemas de generación de planes de evacuación con el objetivo de comenzar a conocer cómo se modela en cada sistema. En el caso de DLV utilizamos el front-end DLVK el cual está diseñado para modelar problemas de planificación.
Comenzamos definiendo cada uno de los elementos de *AS Planning*. Seguido de esto realizamos el modelado del escenario con cada uno de los sistemas con la finalidad de comparar las características de cada uno y generar los planes que corresponden a los *answer sets*.
En el capítulo 5 ampliaremos el dominio de nuestro modelo basándose en los elementos propuestos en el capítulo 3. Modelaremos la capacidad en las poblaciones para poder recibir un número limitado de vehículos en un mismo instante de tiempo, principalmente en aquéllas que son consideradas como refugios. Consideraremos el número de habitantes en las poblaciones ubicadas dentro de la zona de riesgo. Los autobuses tendrán capacidad limitada para evacuar a las personas de las poblaciones ubicadas en la zona de riesgo. Además los autobuses tendrán que ir y regresar de la zona de riesgo a los refugios y viceversa, hasta que el número de habitantes sea igual a cero en las poblaciones en riesgo. Tomaremos en cuenta la capacidad y el sentido de los caminos

que sirven como ruta de evacuación. Se modelará la concurrencia, es decir, que un mismo autobús no pueda realizar dos acciones diferentes al mismo tiempo. Todo esto con el objetivo de trabajar con modelos más apegados al mundo real con respecto a [2] y ampliar lo propuesto en dicho trabajo de tesis.

Capítulo 5

Ampliando el dominio del problema con SMODELS

En el capítulo 4 trabajamos con un dominio en el cual solamente se consideraba un único fluente el cual indicaba la posición del autobús y una sola acción la cual indica que el autobús viaja de un punto a otro. No se tomaban en cuenta el sentido de los caminos, tampoco se consideraba la capacidad en los caminos ni en las poblaciones, la capacidad de los autobuses es ilimitada y no se modela el número de habitantes en las poblaciones en la zona en riesgo. De manera general las condiciones del dominio en el capítulo 4 son muy ideales.

En este capítulo se presenta el modelado de los elementos del dominio definido en el capítulo 3. Modelaremos la capacidad en las poblaciones y los caminos para recibir un número limitado de vehículos en un mismo instante de tiempo. Consideraremos el número de habitantes en las poblaciones ubicados dentro de la zona de riesgo. Los autobuses tendrán una capacidad limitada para evacuar a las personas que se encuentran en las poblaciones en riesgo. Además los autobuses tendrán que ir y regresar de la zona de riesgo a los refugios, y viceversa, hasta que el número de habitantes sea igual a cero en las poblaciones en riesgo. Se tomará en cuenta el sentido de los caminos que en determinado momento pueden servir como ruta de evacuación. Se modelará la concurrencia

entre acciones, en la cual las acciones ocurrirán de manera concurrente en un mismo instante de tiempo siempre y cuando sean realizadas por diferentes autobuses. En otras palabras un mismo autobús no podrá realizar dos o más acciones diferentes al mismo tiempo, lo cual sabemos por sentido común que no es posible.

5.1. Modelado del dominio en *AS Planning*

Como se mencionó anteriormente, el dominio que modelaremos en este capítulo considera el sentido y la capacidad de los caminos, así como el número de habitantes en las poblaciones que se ubican en la zona de riesgo. Para esto se definen nuevos fluentes que nos ayuden a modelar el sentido de los autobuses y el número de habitantes que se encuentran en las poblaciones en riesgo. También definiremos acciones que nos indiquen cuando un autobús está regresando hacia la zona de riesgo o está en una población en riesgo y subiendo a más personas para ser evacuadas. Definiremos reglas para indicar la capacidad limitada de las poblaciones para recibir a un número determinado de autobuses en un mismo instante. Modelaremos la concurrencia entre acciones, en la cual las acciones ocurrierán de manera concurrente en un mismo instante de tiempo siempre y cuando sean realizadas por diferentes autobuses. Además, el escenario modelado para este caso también tendrá más poblaciones, autobuses y caminos con respecto al escenario modelado en el capítulo 4.

Ejemplo 5.1. Para ejemplificar el dominio tenemos una red que consta de 4 poblaciones. La población 1 y la población 4 se encuentran en la zona de riesgo. La población 3 es un refugio. Existe un autobús en la población 1 y en la población 4 los cuales tienen como meta u objetivo llegar al refugio. El número de habitantes en cada población en riesgo es de 200. La población 1 está conectada con la población 2 y ésta última

a la población 3. Por otra parte, la población 4 está conectada con la población 2. La población 1, 3 y 4 tienen capacidad para recibir a dos autobuses, mientras que la población 2 sólo puede recibir 1 autobús. Los caminos de 1 a 2 y de 4 a 2 tienen capacidad para dos autobuses, mientras que el camino de 2 a 3 tienen capacidad para 1. Todos los caminos pueden ser utilizados para ir hacia el refugio o para regresar a la zona de riesgos, es decir en ambos sentidos. El resultado de estas nuevas condiciones generan la figura 5.1.

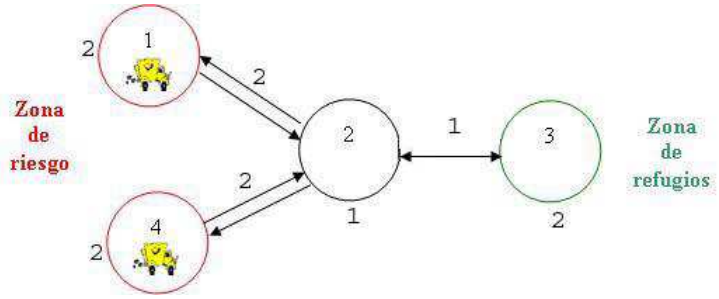

Figura 5.1: Red de poblaciones y caminos.

- ***Background knowledge.*** Para el *background knowledge* se retoma el implementado para el ejemplo 4.1., solamente agregamos una nueva población y nuevos caminos. Además, ahora para cada población se modela su capacidad. También para el caso de los caminos se modela la capacidad y el sentido de cada uno. En este escenario tenemos dos autobuses cada uno de ellos ubicado en una población en riesgo distinta.

  %Se definen los nodos que representan a las poblaciones y su capacidad.

  %node_cap(nodo, capacidad).

  ```
  node_cap(1,2).  node_cap(2,1).  node_cap(3,2).  node_cap(4,2).
  ```

  %Se declaran cuáles son las poblaciones en riesgo.

```
risk_node(1).  risk_node(4).
%Se define la población que es considerada como refugio.
refuge_node(3).
%Se declaran los caminos entre las poblaciones y la capacidad de cada uno.
%Conjunto de segmentos que permiten evacuar hacia la zona de refugios.
%going_way(población inicial, población final, capacidad).
going_way(1,2,2).  going_way(2,3,1).  going_way(4,2,2).
%Conjunto de segmentos que permiten regresar a la zona de riesgo.
%return_way(población inicial, población final, capacidad).
return_way(2,1,2).  return_way(3,2,1).  return_way(2,4,2).
%Se definen los autobuses.
%bus(identificador del autobús).
bus(a1).  bus(a2).
```

Una vez representado el escenario, a continuación se modelan los elementos de *AS Planning*: fluentes, acciones, condiciones de ejecución de las acciones, consecuencias de ejecución de las acciones, condiciones iniciales y metas que nos ayuden a representar nuestro nuevo dominio. Retomaremos el fluente `position` y la acción `evacuate` definidos en el capítulo 4.

- **Fluentes.** Los nuevos fluentes son:

  num_p(N,P)

  Con este fluente se indica el número de habitantes en una población en riesgo, donde P es el número de habitantes en la población en riesgo N.

  direction(B,D)

Con este fluente se indica la dirección D del autobús B. Si D vale 1, quiere decir que la dirección de la trayectoria del autobús es hacia el refugio y si vale 0 la dirección es hacia la zona en riesgo.

status(B,E)

Indica el estado E del autobús B. El estado puede tener los valores de ocupado o vacío.

- **Condiciones iniciales (Estado inicial)**. La nuevas condiciones iniciales son que el autobús `a1` se encuentra en la población en riesgo 1, la cual cuenta con 200 habitantes. El sentido del autobús `a1` es hacia la zona de refugios y además inicialmente se encuentra vacío. El autobús `a2` se encuentra en la población en riesgo 4, la cual cuenta con 200 habitantes. El sentido del autobús `a2` es hacia la zona de refugios y además inicialmente se encuentra vacío.

```
initially:
    position(a1,1).
    position(a2,4).
    direction(a1,1).
    direction(a2,1).
    num_p(1,200).
    num_p(4,200).
    status(a1,vacio).
    status(a2,vacio).
```

- **Condiciones finales (Estado final o meta)**. La nueva meta es que tanto el autobús `a1` como el `a2` lleguen al refugio 3. Además el número de habitantes en las poblaciones en riesgo tiene que ser 0. Ya que el objetivo es que no existan

personas en las poblaciones ubicadas dentro de la zona de riesgo.

```
goal:
    position(a1,3).
    num_p(1,0).
    position(a2,3).
    num_p(4,0).
```

- **Acciones.** Se definieron tres nuevas acciones las cuales son:

  `load(B,N)`

  El autobús B carga a las personas de la población en riesgo N para ser evacuadas.

  `unload(B,N)`

  Indica que las personas evacuadas por el autobús B son dejadas en la población refugio N. Con esto ahora el autobús B se encuentra vacío.

  `return(B,I,F)`

  Indica que el autobús B se encuentra regresando a la zona de riesgo partiendo de la población I a la población F. Esto en caso de que la meta no haya sido cumplida y haya más habitantes en la zona de riesgo.

- **Condiciones de ejecución de las acciones.** Las condiciones para cada acción definida anteriormente son las siguientes:

  `load(B,N) if status(B,empty), risk_node(N).`

  La acción `load` sólo se permite si el autobus B se encuentra vacío `empty`, es decir, en condiciones para que la gente pueda ser transportada. Además sólo es posible en aquellas poblaciones que son consideradas como poblaciones en riesgo.

```
unload(B,N) if status(B,busy), refuge_node(N).
```

La acción `unload` sólo es posible si la posición del autobús B es un nodo refugio y además el autobús tiene personas abordo, es decir, está ocupado `busy`.

```
return(B,I,F) if position(B,I), return_way(I,F), direction(B,0).
```

La acción `return` es posible si el autobús B se encuentra en la población I. Además, existe un camino que conecte a la población I con la población F y la dirección del autobús es hacia la zona de riesgo.

- **Consecuencias de ejecución de las acciones.** La primera consecuencia de la acción `load` es que se realiza la resta del número actual de habitantes en la población `P_actual` menos el número de personas que puede transporar el autobús `Capacidad`. Como consecuencia se tiene un nuevo número de habitantes `P_new`.

```
num_p(N, P_new = P_actual - Capacidad) after load(B,N).
```

Otra consecuencia de la acción `load` es que el número de habitantes en la población en riesgo ya no es P_actual.

```
-num_p(N,P_actual) after load(B,N).
```

La tercera consecuencia de la acción `load` es que la dirección del autobús cambia hacia la zona de refugios.

```
direction(B,1) after load(B,N).
```

La cuarta consecuencia de la acción `load` es que la dirección no es hacia la zona de riesgo.

```
-direction(B,0) after load(B,N).
```

La quinta consecuencia es que el estado del autobús es ahora ocupado `busy`.

```
status(B,busy) after load(B,N).
```

La última consecuencia de la acción `load` es que el autobús ya no está vacío `empty`.

```
-status(B,empty) after load(B,N).
```

Ahora presentamos las consecuencias de la acción `unload`. La primera consecuencia es que el autobús ya no está ocupado `busy`.

```
-status(B,busy) after unload(B,N).
```

La segunda consecuencia de la acción `unload` es que el autobús está vacío `empty`.

```
status(B,empty) after unload(B,N).
```

Finalmente las consecuencias de la acción `return`. La primera consecuencia es que la nueva posición del autobús B es la población F.

```
position(B,F) after return(B,I,F).
```

La segunda consecuencia de la acción `return` es que la posición del autobús B ya no es la población I.

```
-position(B,I) after return(B,I,F).
```

Con el objetivo de ampliar lo propuesto en [2] y basándose en lo definido en [10] y [1] otro elemento importante para ser modelado es la concurrencia como se muestra a continuación:

- **Manejo de concurrencia.** Aunque el manejo de la concurrencia entre las acciones no es necesariamente uno de los elementos mencionados de *AS Planning* es de gran importancia el modelado de dicha característica, porque por sentido común sabemos que un objeto, persona o cosa no puede realizar dos acciones diferentes al mismo tiempo.

  En nuestro caso la concurrencia se refiere al hecho de que si existe un autobús B realizando cualquier acción de las antes mencionadas y representada como A1, entonces este mismo autobús no puede realizar una acción distinta denotada como A2 en el mismo instante de tiempo T. Esto se modela con la siguiente regla:

```
no_ocurre(accion(B,A2),T) if ocurre(accion(B,A1),T), A1<>A2.
```

Otro aspecto a modelar es la capacidad de los caminos y poblaciones.

- **Representación de capacidad en las poblaciones y caminos.** La capacidad de las poblaciones y los caminos pueden variar de acuerdo a las características del escenario. Aquí sólo se muestra cómo se modela para capacidad igual a uno pero de manera semejante se pueden agregar reglas para diferentes capacidades. Como veremos en las siguientes secciones.

  - Para el caso de las poblaciones, si un autobús B1 trata de evacuar a una población F que tiene capacidad para sólo un autobús, no podrá hacerlo si ya está un autobús B2 en la misma población F en el mismo instante de tiempo T. Donde B1 y B2 son diferentes.

    ```
    no_ocurre(evacuar(B1,I,F),T) if
                    capacidad(F,1), ocurre(position(B2,F),T), B1<>B2.
    ```

Si un autobús B1 trata de evacuar a una población F que tiene capacidad para sólo un autobús no podrá hacerlo si un autobús B2 está haciendo la misma acción hacia la misma población F en el mismo instante de tiempo T. Donde B1 y B2 son diferentes.

```
no_ocurre(evacuar(B1,I,F),T) if
                capacidad(F,1), ocurre(evacuar(B2,I,F),T), B1<>B2.
```

El caso inverso también ocurre si un autobús B1 trata de regresar a una población F que tiene capacidad para sólo un autobús no podrá hacerlo si ya está un autobús B2 en la misma población F en el mismo instante de tiempo T. Donde B1 y B2 son diferentes.

```
no_ocurre(regresar(B1,I,F),T) if
                capacidad(F,1), ocurre(position(B2,F),T), B1<>B2.
```

Ahora bien, si un autobús B1 trata de regresar a una población F que tiene capacidad para sólo un autobús no podrá hacerlo si un autobús B2 está haciendo la misma acción hacia la misma población F en el mismo instante de tiempo T. Donde B1 y B2 son diferentes.

```
no_ocurre(regresar(B1,I,F),T) if
                capacidad(F,1), ocurre(regresar(B2,I,F,T), B1<>B2.
```

- Para el caso de los caminos si un autobús B1 trata de evacuar por un camino que va de I a F el cual sólo tiene capacidad para un autobús no podrá hacerlo si un autobús B2 está realizando la misma acción por el mismo camino en el mismo instante de tiempo T. Donde B1 y B2 son diferentes.

  ```
  no_ocurre(evacuar(B1,I,F),T) if
  ```

```
camino_ida(I,F,1), ocurre(evacuar(B2,I,F), B1<>B2.
```

Por otro lado, si un autobús B1 trata de regresar por un camino que va de I a F el cual sólo tiene capacidad para un autobús no podrá hacerlo si un autobús B2 está realizando la misma acción por el mismo camino en el mismo instante de tiempo T. Donde B1 y B2 son diferentes

```
no_ocurre(regresar(B1,I,F),T) if
        camino_regreso(I,F,1), ocurre(regresar(B2,I,F), B1<>B2.
```

Como se puede apreciar con el modelado del ejemplo anterior, las poblaciones, los caminos, el número de autobuses, los fluentes, las acciones con sus consecuencias y condiciones de ejecución, pueden ser tan extensos como el problema lo requiera, sólo basta agregar dichas modificaciones a lo que se tiene sin necesidad de tener que cambiar el modelo inicial. Esta es una de las grandes ventajas que tiene ASP.

5.2. Modelado en SMODELS

En esta sección se presenta la implmentación del escenario mostrado en el ejemplo 5.1. utilizando SMODELS. La implementación retomará el modelado en SMODELS realizado en la sección 4.2 por lo que no se comenzará desde cero, basta simplemente con agregar las nuevas características del dominio definidas en este capítulo.

```
%José Miguel Leyva Cruz, 18/11/2006 UTM
%ejemplo2.sm

%Se declara una constante númerica n la cual representa el tiempo
```

```
%máximo posible que se requiere para alcanzar la meta. En cada
%instante de tiempo se ejecuta una acción.
const n=8.
time(1..n).

% BACKGROUND KNOWLEDGE
% Conjunto de poblaciones. Donde cada población es un nodo.
%node_cap(población, capacidad).
node_cap(1,2).  node_cap(2,1).  node_cap(3,2).  node_cap(4,2).

%Lista de nodos considerados como refugios.
refuge_node(3).

%Conjunto de caminos y capacidad de cada camino.
%Conjunto de caminos que permiten evacuar hacia la zona de refugios.
%going_way(población inicial, población final, capacidad).
going_way(1,2,2).  going_way(4,2,2).  going_way(2,3,1).

%Conjunto de caminos que permiten regresar a la zona de riesgo.
%return_way(población inicial, población final, capacidad)
return_way(2,1,2).  return_way(2,4,2).  return_way(3,2,1).

%Valores para indicar la dirección que llevan los autobuses.
%Si el valor es 1 se dirigen hacia la zona de refugios.
%Si el valor es 0 se dirigen hacia la zona de riesgo.
%dir_value(valor).
```

```
dir_value(0).  dir_value(1).

%Se definen los autobuses.
%bus(identificador del autobús).
bus(a1).  bus(a2).

%Condiciones iniciales. Posición inicial del autobús a1 es la
%población 1 y la dirección es hacia el refugio. La posición del
%autobús a2 es en la población 4 y la dirección es hacia el refugio.
initially(position(a1,1)).       initially(direction(a1,1)).
initially(position(a2,4)).       initially(direction(a2,1)).

%Condiciones finales o meta. La posición final del autobús a1 es la
%población 1 y la dirección es hacia la zona de riesgo. La posición del
%autobús a2 es en la población 4 y la dirección es hacia la zona de riesgo.
finally(position(a1,1)).
finally(direction(a1,0)).
finally(position(a2,4)).
finally(direction(a2,0)).

%Se verifica que la meta sea alcanzada.
%No se quiere que no haya un plan, es decir, se quiere que haya un plan.
:- not plan.

%Existe un plan si se llega a la meta a lo más en n pasos.
plan :- goal(n).
```

```
%La meta se cumple en el tiempo T si no hay evidencia de que la
%meta no haya sido cumplida.
goal(T) :- time(T),
       not not_goal(T).

%La meta no se cumple en el tiempo T si no hay evidencia de que se han
%cumplido las condiciones finales (finally).
not_goal(T) :- time(T),
       literal(X),
       finally(X),
       not hold(X,T).

%%%FLUENTES
%Indica que la posición del autobús B1 es el nodo N.
fluent(position(B1,N)) :- bus(B1), node_cap(N,C).

%Indica la dirección D del autobús B1.
%Si D es igual a 1 se dirige hacia la zona de refugios.
%si D es igual a 0 se dirige hacia la zona en riesgo.
fluent(direction (B1,D)):-bus(B1), dir_value(D).

%%%ACCIONES
%evacuate: el autobús B1 viaja de la población I a la
%población F siempre que haya un camino de ida de I-F.
action(act(B1,evacuate(I,F))) :- going_way(I,F,C), bus(B1).
```

```
%return: el autobús B1 viaja de la población I a la
%población F siempre que haya un camino de regreso de I-F.
action(act(B1,return(I,F))) :- return_way(I,F,C), bus(B1).

%turn: el autobús B1 gira en el nodo refugio N para regresar a la
%zona de riesgo. Esta acción equivale a la acción definida en la
%sección 5.1. como unload. Dicho nombre se cambió dado que en SMODELS
%no fue fácil modelar el número de habitantes en las poblaciones en riesgo.
%Más adelante se dará una explicación más detallada de este aspecto.
action(act(B1,turn(N))):- bus(B1), refuge_node(N).

%%%CONSECUENCIAS DE EJECUCIÓN DE LAS ACCIONES
%La primera consecuencia de la acción evacuate es que la nueva posición
%del autobús B1 es la población F.
caused(position(B1,F),act(B1,evacuate(I,F))):-
                                              bus(B1), going_way(I,F,C).
%La segunda consecuencia de la acción evacuate es que la posición
%del autobús B1 ya no es la población I.
caused(neg(position(B1,I)),act(B1,evacuate(I,F))):-
                                              bus(B1), going_way(I,F,C).

%La primera consecuencia de la acción return es que la nueva posición
%del autobús B1 es F.
caused(position(B1,F),act(B1,return(I,F))):-
                                          bus(B1), return_way(I,F,C).
```

```
%La segunda consecuencia de la acción return es que la posición del
%autobús B1 ya no es la población I.
caused(neg(position(B1,I)),act(B1,return(I,F))):-
                                                bus(B1), return_way(I,F,C).

%La primera consecuencia de la acción turn es que el sentido del
%autobús B1 cambia a 0, es decir, el autobús se encuentra en dirección
%hacia la zona en riesgo.
caused(direction(B1,0),act(B1,turn(N))):- bus(B1), refuge_node(N).
%La segunda consecuencia de la acción turn es que el valor de la
%dirección del autobús B1 ya no es 1, es decir, el autobús ya no se dirige
%hacia la zona de refugios.
caused(neg(direction(B1,1)),act(B1,turn(N))):- bus(B1), refuge_node(N).

%%%CONDICIONES DE EJECUCIÓN DE LAS ACCIONES
%La acción evacuate no es posible si el autobús B1 no está en la
%población I y además debe existir un camino que una a la
%población I con la población F.
noaction_if(act(B1,evacuate(I,F)),neg(position(B1,I))):-
                                                bus(B1), going_way(I,F,C).
%La acción evacuate no es posible si la dirección del autobús
%B1 no es 1, es decir, hacia la la zona de refugios.
noaction_if(act(B1,evacuate(I,F)),neg(direction(B1,1))):-
                                                bus(B1), going_way(I,F,C).

%La acción return no es posible si el autobús B1 no está en la
```

```
%población I. Además debe existir un camino de regreso que una a la
%población I con la población F.
noaction_if(act(B1,return(I,F)),neg(position(B1,I))):-
                                                bus(B1), return_way(I,F,C).
%La acción evacuate no es posible si la dirección del autobús
%B1 no es 1, es decir, hacia la zona de refugios.
noaction_if(act(B1,return(I,F)),neg(direction(B1,0))):-
                                                bus(B1), return_way(I,F,C).

%La acción turn no es posible si el autobús B1 no se encuentra en
%un nodo refugio N.
noaction_if(act(B1,turn(N)),neg(position(B1,N))):-bus(B1), refuge_node(N).

%%%RESTRICCIONES DE EJECUCIÓN
%Recordemos que las restricciones de ejecución siempre se colocan para
%modelar cualquier problema de planificación en SMODELS.
%La acción A no es ejecutable en el tiempo T si las condiciones de
%ejecución para A no se cumplen.
not_executable(A,T) :-
            action(A),
            time(T),
            noaction_if(A,F),
            hold(F,T).

%La acción A es ejecutable en el tiempo T si no hay evidencia
%de que A no pueda ser ejecutada.
```

```
executable(A,T) :-
        action (A),
        time (T),
        not not_executable(A,T).

% Es posible ejecutar la acción A en el tiempo T si no hay evidencia
%de que la meta no haya sido cumplida.
possible(A,T) :-
        action(A),
        executable(A,T),
        time(T),
        not goal(T).

%El valor del fluente F cambia su valor en el tiempo T+1 si la
%acción A afecta a F. Además la acción A es ejecutable y está
%ocurriendo en el tiempo T.
hold(F,T+1) :- literal(F),
        time(T),
        T<n,
        action(A),
        executable(A,T),
        occurs(A,T),
        caused(F,A).

%El fluente F siempre se cumple en el tiempo 1 si se define
%como condición inicial. Esto se cumple para todos los fluentes
```

```
%definidos como initially.
hold(F,1) :- literal(F), initially(F).

%El fluente F no se cumple en el tiempo 1 si no está definido
%como condición inicial. Esto se cumple para todos los fluentes
%que no están definidos como initially.
hold(neg(F), 1) :- fluent(F), not hold(F,1).

%Si F es un fluente, su valor contrario es -F.
%Recordemos que la negación de un fluente se escribe con el símbolo -.
contrary(F, neg(F)) :- fluent(F).

%Si -F es un fluente, su valor contrario es F.
contrary(neg(F), F) :- fluent(F).

%La literal G es un fluente.
literal(G) :- fluent(G).

%La negación de la literal G también es un fluente.
literal(neg(G)) :- fluent(G).

%La literal F mantiene su valor en el tiempo T+1 si no hay
%evidencia de que cambie a su valor contrario G. Esto se conoce como
%la ley de la inercia, un fluente mantiene su valor hasta que no haya
%evidencia de que éste haya cambiado.
hold(F, T+1) :-literal(F),
```

```
        literal(G),
        contrary(F,G),
        time(T),
        T<n,
        hold(F,T),
        not hold(G, T+1).
%%%FIN DE LAS RESTRICCIONES DE EJECUCIÓN.

%La acción A ocurre en el tiempo T si no hay evidencia de que la acción
no
%pueda ocurrir en dicho instante.
occurs(A,T) :-
        action(A),
        time(T),
        possible(A,T),
        not not_occurs(A,T).

%La acción A ocurre en el tiempo T si se verifica que A es posible y no
%existe evidencia de que no pueda ocurrir en dicho tiempo.
occurs(A,T) :-
        action(A),
        time(T),
        possible(A,T),
        verify(A,T),
        not not_occurs(A,T).
```

```
%Se definen subacciones internas. Esto se utilizará más
%adelante para el manejo de la concurrencia.
sub_action(evacuate(I,F)) :- going_way(I,F,C).
sub_action(return(I,F)) :- return_way(I,F,C).
sub_action(turn(N)):- refuge_node(N).

%%%CONCURRENCIA
%El autobús B1 no puede realizar la acción A1 en el tiempo T si ya está
%realizando otra acción A2. Además A1 y A2 son diferentes.
not_occurs(act(B1,A1),T) :-
       action(act(B1,A1)), sub_action(A1),bus(B1),
       action(act(B1,A2)), sub_action(A2),
       time(T),
       occurs(act(B1,A2),T),
       neq(A1,A2).

%%%CAPACIDAD EN CAMINOS
%El autobús B1 no podrá evacuar por el camino I-F si éste tiene
%capacidad para un sólo autobús y el autobús B2 está evacuando
%en el mismo tiempo T por el camino I-F. Además B1 y B2 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
       time(T),
       bus(B1),
       going_way(I,F,1),
       occurs(act(B2,evacuate(I,F)),T), bus(B2),
       neq(B1,B2).
```

```
%El autobús B1 no podrá regresar por el camino I-F si éste tiene
%capacidad para un sólo autobús y el autobús B2 está regresando
%en el mismo tiempo T por el camino I-F. Además B1 y B2 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
       time(T),
       bus(B1),
       return_way(I,F,1),
       occurs(act(B2,return(I,F)),T), bus(B2),
       neq(B1,B2).

%El autobús B1 no podrá evacuar por el camino I-F si éste tiene
%capacidad para dos autobuses y los autobuses B2 y B3 están evacuando
%en el mismo tiempo T por el camino I-F. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
       time(T),
       bus(B1),
       going_way(I,F,2),
       occurs(act(B2,evacuate(I,F)),T), bus(B2),
       occurs(act(B3,evacuate(I,F)),T), bus(B3),
       neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá regresar por el camino I-F si éste tiene
%capacidad para dos autobuses y los autobuses B2 y B3 están regresando
%en el mismo tiempo T por el camino I-F. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
```

```
        time(T),
        bus(B1),
        return_way(I,F,2),
        occurs(act(B2,return(I,F)),T), bus(B2),
        occurs(act(B3,return(I,F)),T), bus(B3),
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%%%CAPACIDAD EN LAS POBLACIONES
%El autobús B1 no podrá evacuar hacia la población F si ésta tiene
%capacidad para un autobús y el autobús B2 también está evacuando
%hacia la misma población F en el mismo tiempo T. Además B1 y B2
%son diferentes.
not_occurs(act(B1,evacuate(I1,F)),T):-
        time(T),
        bus(B1),
        node_cap(I1,C1),
        node_cap(F,1),
        occurs(act(B2,evacuate(I2,F)),T), bus(B2), node_cap(I2,C2),
        neq(B1,B2).

%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 también está regresando
%hacia la misma población F en el mismo tiempo T. Además B1 y B2
%son diferentes.
not_occurs(act(B1,return(I1,F)),T):-
        time(T),
```

```
        bus(B1),
        node_cap(I1,C1),
        node_cap(F,1),
        occurs(act(B2,return(I2,F)),T), bus(B2), node_cap(I2,C2),
        neq(B1,B2).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 ya está en la población F
%en el mismo tiempo T. Además B1 y B2 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C1),
        hold(position(B2,F),T),
        node_cap(F,1),
        bus(B2),neq(B1,B2).

%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 ya está en la población F
%en el mismo tiempo T. Además B1 y B2 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C1),
        hold(position(B2,F),T),
        node_cap(F,1),
```

```
        bus(B2),neq(B1,B2).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 también están
%evacuando hacia la misma población F en el mismo tiempo T. Además
%B1, B2 y B3 son diferentes.
not_occurs(act(B1,evacuate(I1,F)),T):-
        time(T),
        bus(B1),
        node_cap(I1,C),
        node_cap(F,2),
        occurs(act(B2,evacuate(I2,F)),T), bus(B2), node_cap(I2,C),
        occurs(act(B3,evacuate(I3,F)),T), bus(B3), node_cap(I3,C),
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 también están
%regresando hacia la misma población F en el mismo tiempo T. Además
% B1, B2 y B3 son diferentes.
not_occurs(act(B1,return(I1,F)),T):-
        time(T),
        bus(B1),
        node_cap(I1,C),
        node_cap(F,2),
        occurs(act(B2,return(I2,F)),T), bus(B2), node_cap(I2,C),
        occurs(act(B3,return(I3,F)),T), bus(B3), node_cap(I3,C),
```

```
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 ya están en la
%población F en el mismo tiempo T. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C),
        hold(position(B2,F),T),
        hold(position(B3,F),T),
        node_cap(F,2),
        bus(B2), bus(B3),
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 ya están en la
%población F en el mismo tiempo T. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C),
        hold(position(B2,F),T),
        hold(position(B3,F),T),
        node_cap(F,2),
        bus(B2), bus(B3),
```

```
      neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 podrá evacuar hacia la población F, la cual se
%encuentra ocupada en su máxima capacidad, si se verifica que en
%el mismo tiempo T algún autobús que ya se encuentra en la población
%F evacua hacia otra población.
verify(act(B1,evacuate(I1,F)),T):-
      time(T),
      bus(B1),
      going_way(I1,F,C1),
      node_cap(F,X),
      occurs(act(B2,evacuate(F,F2)),T), bus(B2),
      going_way(F,F2,C2),
      neq(B1,B2).
%El autobús B1 podrá regresar hacia la población F, la cual se
%encuentra ocupada en su máxima capacidad, si se verifica que en
%el mismo tiempo T algún autobús que ya se encuentra en la población
%F regresa hacia otra población.
verify(act(B1,return(I1,F)),T):-
      time(T),
      bus(B1),
      return_way(I1,F,c1),
      node_cap(F,X),
      occurs(act(B2,return(F,F2)),T), bus(B2),
      return_way(F,F2,c2),
      neq(B1,B2).
```

Para poder ejecutar el programa anterior se escribe la siguiente instrucción en la línea de comandos.

```
lparse < ejemplo2.sm | smodels 0
```

Los planes generados son únicamente dos, los cuales corresponden a los *answer sets*. Como se puede ver los dos autobuses salen cada uno de su población en riesgo, llegan al refugio y regresan a la población en riesgo para evacuar a más personas. Las tablas 5.1 y 5.2 muestran los planes. La palabra *no_action* indica que durante ese instante de tiempo un autobús no está realizando alguna acción.

TIEMPO	ACCIONES	
1	evacuate(a1, 1, 2)	no_action
2	evacuate(a1, 2, 3)	evacuate(a2, 4, 2)
3	turn(a1, 3)	evacuate(a2, 2, 3)
4	return(a1, 3, 2)	turn(a2, 3)
5	return(a1, 2, 1)	turn(a2,3)
6	no_action	return(a2, 3, 2)
7	no_action	return(a2, 2, 4)

Tabla 5.1: *Answer set* generado por SMODELS

TIEMPO	ACCIONES	
1	no_action	evacuate(a2, 4, 2)
2	evacuate(a1, 1, 2)	evacuate(a2, 2, 3)
3	evacuate(a1, 2, 3)	turn(a2, 3)
4	turn(a1, 3)	return(a2, 3, 2)
5	turn(a1, 3)	return(a2, 2, 4)
6	return(a1, 3, 2)	no_action
7	return(a1, 2, 1)	no_action

Tabla 5.2: *Answer set* generado por SMODELS

5.3. Contribución

A pesar de que SMODELS es un sistema de propósito general resultó ser una muy buena herramienta para modelar problemas de planificación. En este caso lo utilizamos para modelar problemas de generación de planes de evacuación. SMODELS posee un conjunto de instrucciones que nos permitió modelar los elementos principales de *AS Planning*: condiciones iniciales, condiciones finales o meta, acciones, condiciones y consecuencias de ejecución de las acciones. Además SMODELS permite modelar características tales como: concurrencia, restricciones, capacidad en los caminos y poblaciones, etc. con lo cual hace que los escenarios que se modelan estén más apegados a la realidad consiguiendo con esto uno de los objetivos de este trabajo de tesis: ampliar lo propuesto en [2].

SMODELS maneja de manera clara y sencilla la concurrencia entre acciones lo que permite adecuar dicha característica de acuerdo al dominio del problema que se quiera modelar. Se pudo modelar perfectamente las capacidades en poblaciones y caminos. Es decir, que en una población o camino no existan más autobuses si alguno de éstos se encuentran ya ocupados a su capacidad máxima y permitir que sean ocupados por más autobuses sólo en el instante en que sean desocupados por alguno de los autobuses que ya están en ellos. Esto se logró gracias a que SMODELS permite comparar acciones contra acciones antes de que sean ejecutadas. Por ejemplo, si el autobús `B1` va a evacuar hacia la población `F` la cual tiene capacidad para un solo autobús, no podrá hacerlo si otro autobús `B2` está realizando también la acción evacuar hacia la misma población `F` y con esto la población ya no tendrá más capacidad para recibir a `B1`. Otro aspecto de SMODELS es que también permite comparar acciones durante su ejecución. Por ejemplo, si en un tiempo `T` el autobús `B1` va a evacuar hacia a la población `F` la cual está ocupada en su máxima capacidad podrá hacerlo sólo si en ese mismo tiempo `T` el

autobús B2, que ya se encuentra en F, evacúa de F hacia otra población y con esto F puede recibir a B1. Básicamente verificamos las acciones que se están ejecutando lo cual es diferente a las condiciones de ejecución ya que en éstas se verifica que los fluentes cumplan con las condiciones necesarias antes del comienzo de la ejecución de una acción.

En SMODELS no fue fácil modelar el número de habitantes en las poblaciones en riesgo. Esto porque al tratar de modelar dicha característica SMODELS no permitió trabajarlo de una manera sencilla, es decir, que simplemente se definiera una operación en la cual el nuevo número de habitantes en la población en riesgo (P_new) fuera igual a la resta del número actual de habitantes en la población en riesgo (P_actual) menos la capacidad del autobús (Capacidad) la cual indica el número máximo de habitantes que pueden ser transportados. La operación anterior queda definida como:

$$\texttt{P\_new} = \texttt{P\_actual} - \texttt{Capacidad}$$

En SMODELS modelamos la operación anterior con las siguientes reglas:

```
caused(num_p(N,P-1),act(B1,load(N))):- bus(B1), risk_node(N), people(P).
caused(neg(num_p(N,P)),act(B1,load(N))):- bus(B1), risk_node(N), people(P).
```

Esto hace que el sistema genere varios fluentes verdaderos para cada valor de acuerdo al número de habitantes que se encuentran en la población en riesgo lo cual sabemos por sentido común no es posible, ya que sólo debe cumplirse como verdadero el fluente que represente el número actual de habitantes que se encuentran en la población en riesgo. A continuación se presenta un ejemplo donde explicaremos más detalladamente el problema. Posteriormente propondremos una nueva solución.

Ejemplo 5.2. Supongamos que se tiene una red la cual consta de 3 poblaciones. La población en riesgo 1 tiene 3 habitantes y en ella se encuentra un autobús que tiene capacidad para evacuar 1 habitante. La meta es que el autobús llegue a la población 3

la cual es un refugio y que el número de habitantes en la población en riesgo sea 0. La figura 5.2 muestra el escenario descrito anteriormente.

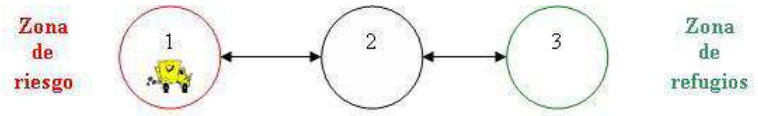

Figura 5.2: Red de poblaciones y caminos.

Retomando los fluentes y las acciones definidos en el ejemplo 5.1. solamente modelamos el nuevo fluente que representa el número de habitantes `num_p(N,P)`, después la acción `load(N)` y finalmente la acción `unload(N)`. Notesé que la resta que nosotros indicamos anteriormente se ve en las consecuencias de ejecución de las acciones y que en este caso involucra a la acción `load(N)`.

A continuación se presenta el código que modela el fluente y acciones mencionadas anteriormente, tomando como base el código presentado para el ejemplo 5.1.

```
%Se declara una constante númerica.
const p=3.
%Se declara la variable que representa el número de habitantes.
people(0..3).

%Fluente que representa el número de habitantes P en la población
%en riesgo N.
fluent(num_p(N,P)):-risk_node(N),people(P).

%Condiciones iniciales. Posición inicial del autobús a1 es la
%población 1 la cual tiene 3 habitantes.
```

```
initially(position(a1,1)).
initially(num_p(1,3)).

%Condiciones finales o metas. La posición final del autobús a1 es la
%población 3 y el número de habitantes en la población 1 tiene que ser 0.
finally(position(a1,3)).
finally(num_hab(1,0)).

%%ACCIONES.
%Acción que indica que el autobús B1 está cargando a los habitantes
%de la población en riesgo N.
action(act(B1,load(N))):-bus(B1),risk_node(N).

%Indica que las personas evacuadas por el autobús B son dejadas en la
%población refugio N. Con esto ahora el autobús B se encuentra vacío.
action(act(B1,unload(N))):- bus(B1), refuge_node(N).

%%CONDICIONES DE EJECUCIÓN DE LAS ACCIONES.
%La acción load no ocurre si el autobús B1 no está en la población
%N la cual es una población en riesgo.
noaction_if(act(B1,load(N)),neg(position(B1,N))) :- bus(B1), risk_node(N).
%La acción load no ocurre si el autobús B1 no está vacío.
noaction_if(act(B1,load(N)),neg(status(B1,empty))) :- bus(B1), risk_node(N).

%La acción unload no ocurre si el autobús B1 no está en la población
%N la cual es una población refugio.
```

```
noaction_if(act(B1,unload(N)),neg(position(B1,N))) :- bus(B1), refuge_node(N).
%La acción unload no ocurre si el autobús B1 no está ocupado.
noaction_if(act(B1,unload(N)),neg(status(B1,busy))) :- bus(B1), refuge_node(N).

%%CONSECUENCIAS DE EJECUCIÓN DE LAS ACCIONES.
%La primera consecuencia de la acción load es disminuir en 1 el número
%de habitantes P de la población en riesgo N. Se disminuye en 1 porque es
%la capacidad del autobús.
caused(num_p(N,P-1),act(B1,load(N))):- bus(B1), risk_node(N), people(P).
%La segunda consecuencia de la acción load es que el número
%de habitantes de la población en riesgo N ya no es P.
caused(neg(num_p(N,P)),act(B1,load(N))):- bus(B1), risk_node(N), people(P).

%La primera consecuencia de la acción unload es que la dirección
%del autobús B1 es hacia la zona de riesgo.
caused(direction(B1,0),act(B1,unload(N))):- bus(B1), refuge_node(N).
%La segunda consecuencia de la acción unload es que la dirección
%del autobús B1 ya no es hacia la zona de refugios.
caused(neg(direction(B1,1)),act(B1,unload(N))):- bus(B1), refuge_node(N).
%La tercera consecuencia de la acción unload es que el estado del
%autobús B1 es vacio.
caused(status(B1,empty),act(B1,unload(N))):- bus(B1), refuge_node(N).
%La cuarta consecuencia de la acción unload es que el estado del
%autobús B1 ya no es lleno.
caused(neg(status(B1,full)),act(B1,unload(N))):- bus(B1), refuge_node(N).
```

Ejecutando el programa anterior se generan varios planes los cuales corresponden a los *answer sets.* La tabla 5.3 muestra uno de ellos. Además de la secuencia de acciones se muestran los cambios en el fluente `num_p` para ver mejor el problema.

TIEMPO	ACCIONES	FLUENTE `num_p`	FLUENTE `position`
1	load(a1,1)	num_p(1,3) neg(num_p(1,0)) neg(num_p(1,1)) neg(num_p(1,2))	position(a1,1) neg(position(a1,2)) neg(position(a1,3))
2	evacuate(a1, 1, 2)	num_p(1,2) num_p(1,1) num_p(1,0) neg(num_p(1,3)) neg(num_p(1,2)) neg(num_p(1,1)) neg(num_p(1,0))	position(a1,1) neg(position(a1,2)) neg(position(a1,3))
3	evacuate(a1, 2, 3)	num_p(1,2) num_p(1,1) num_p(1,0) neg(num_p(1,3)) neg(num_p(1,2)) neg(num_p(1,1)) neg(num_p(1,0))	position(a1,2) neg(position(a1,0)) neg(position(a1,3))

Tabla 5.3: *Answer set* generado por SMODELS

Analizando los resultados de la tabla 5.3 podemos ver lo siguiente:

Tiempo 1. El autobús `a1` realiza la acción `load(a1,1)`. Se indica que la posición del autobús `a1` es la población 1, por lo que se niega que esté en la población 2 o 3. El número de habitantes en la población 1 es 3, por lo que también se niega que el número de habitantes sean 2, 1, 0. Estos valores corresponden básicamente a las condicones iniciales definidas para este ejemplo.

Tiempo 2. Las consecuencias de la acción `load(a1,1)` cambian el estado del mundo. Recordemos que las consecuencias de ejecución de la acción `load(N)` están modeladas por las siguientes líneas de código:

```
caused(num_p(N,P-1),act(B1,load(N))):- bus(B1), risk_node(N), people(P).
caused(neg(num_p(N,P)),act(B1,load(N))):- bus(B1), risk_node(N), people(P).
```

En este caso lo correcto sería que en el tiempo 2 se cumpliera el fluente `num_p(1,2)` porque al valor del número de habitantes representado por `P` se le resta uno. Recordemos que el autobús tiene capacidad para evacuar a un habitante y el valor de `P` era 3 en el tiempo 1. Así también se espera que se cumpla el fluente `neg(num_p(1,3))`. Con esto decimos que el número actual de habitantes en la población 1 es 2 y ya no 3. Pero como podemos ver en la tabla 5.3 esto no ocurre, ya que la forma en como se modeló el problema hace que SMODELS considere los valores desde 0 hasta el valor actual de P que es 2, cumpliéndose los siguientes fluentes `num_p(1,2), num_p(1,1)` y `num_p(1,0)`. Con lo cual se llega al problema que se mencionó anteriormente: que se cumplan varios fluentes verdaderos para `num_p(N,P)`, lo cual sabemos por sentido común no es posible. También en el mismo instante de tiempo se cumplen los fluentes `neg(num_p(1,2)),neg(num_p(1,1)), neg(num_p(1,0))`. Esto ocasiona que haya contradicción en los fluentes, porque tanto aparecen negados como verdaderos.

Tiempo 3. Notesé que lo que ocurre en el tiempo 1 y en el tiempo 2 nos lleva a tener problemas con los fluentes, por lo que en el tiempo 3 los resultados que se tienen son falsos.

Tiempo 4. Por último se cumplen las condiciones finales definidas para el ejemplo y se da por terminada la ejecución del programa por lo que ya no hay más acciones. Pero como podemos ver en la tabla el plan no es correcto, ya que tomando en cuenta que la población 1 tiene 3 habitantes y que el autobús `a1` puede evacuar de uno en uno, se espera que el autobús `a1` viaje a la población en riesgo otras 2 veces con el objetivo de ir evacuando a los habitantes.

Dado que nuestra primera versión para modelar el número de habitantes, la cual nos resultaba más natural, no funcionó adecuadamente proponemos otra forma de modelar el

número de habitantes en las poblaciones en riesgo utilizando SMODELS. Comenzamos con crear un fluente para cada uno de los valores de acuerdo al número de habitantes que existen en la población en riesgo. Esto nos lleva a modelar varias acciones `load` de acuerdo al número de habitantes que existen en la población en riesgo para que cada que ocurra alguna de ellas, se cumplan como falsos los fluentes que no correspondan al número actual de habitantes en un determinado instante y únicamente se cumpla como verdadero el fluente que corresponda al número actual de habitantes.

Para ejemplificar lo anterior retomemos el escenario de la figura 5.2. Recordemos que la población en riesgo 1 tiene 3 habitantes por lo que se crea un fluente para representar cada valor del número de habitantes en cada una de las poblaciones en riesgo, en este caso para la población 1: `num_p_in_pob_1_3, num_p_in_pob_1_2, num_p_in_pob_1_1` y `num_p_in_pob_1_0`. Se definen las siguientes acciones: `load1(N)` la cual indica que se está evacuando a la primera persona de la población en riesgo `N`, `load2(N)` la cual indica que se está evacuando a la segunda persona y finalmente la acción `load3(N)` para indicar que se está evacuando a la tercera y última persona. A continuación mostramos el código en SMODELS con el nuevo modelado retomando los fluentes y las acciones definidas para el ejemplo 5.1. Los efectos y las condiciones de ejecución de cada acción se explican detalladamente en el código.

```
%CONDICIONES INICIALES.
%El autobús a1 está en la población 1. El estado del autobús
%es vacío y se dirige hacia la zona de refugios. La población
%1 tiene 3 habitantes.
initially(position(a1,1)).
initially(direction(a1,1)).
initially(status(a1,empty)).
```

```
initially(num_p_in_pob_1_3).

%CONDICIONES FINALES O META.
%El autobús a1 está en la población 3.
%El estado del autobús es ocupado. La población 1 tiene 0 habitantes.
finally(position(a1,3)).
finally(status(a1,busy)).
finally(num_p_in_pob_1_0).

%%%FLUENTES.
%Se define un fluente para cada valor permitido de P en la población 1.
%En este caso, como el autobús tiene capacidad para evacuar a 1 persona,
%se definen fluentes con valor de 3, 2, 1 y 0.
fluent(num_p_in_pob_1_3).
fluent(num_p_in_pob_1_2).
fluent(num_p_in_pob_1_1).
fluent(num_p_in_pob_1_0).

%%%ACCIONES.
%Se define la acción load la cual indica que el autobús B1 está evacuando
%a los habitantes. En este caso se definen acciones diferentes para cuando
%el autobús está evacuando a la primera, a la segunda y a la
%tercera persona. Recordemos que son 3 personas y que el autobús tiene
%capacidad para evacuar de uno en uno.
%Se evacua a la primera persona de la población N.
action(act(B1,load1(N))):-bus(B1),risk_node(N).
```

```
%%Se evacua a la segunda persona de la población N.
action(act(B1,load2(N))):-bus(B1),risk_node(N).
%%Se evacua a la tercera persona de la población N.
action(act(B1,load3(N))):-bus(B1),risk_node(N).
%La acción unload indica que las personas evacuadas por el autobús
%B son dejadas en la población refugio N. Con esto ahora el autobús B
%se encuentra vacío.
action(act(B1,unload(N))):- bus(B1), refuge_node(N).

%%CONDICIONES DE EJECUCIÓN DE LAS ACCIONES.
%La acción load no ocurre si la posición del autobús B1 no es
%la población en riesgo N.
noaction_if(act(B1,load1(N)),neg(position(B1,N))):- bus(B1), risk_node(N).
%La acción load1 no ocurre si el número actual de habitantes no
%es 3. Recordemos que load1 evacúa a la primera persona.
noaction_if(act(B1,load1(N)),neg(num_p_in_pob_1_3)):-bus(B1), risk_node(N).

%La acción load2 no ocurre si la posición del autobús B1 no es
%la población en riesgo N.
noaction_if(act(B1,load2(N)),neg(position(B1,N))) :- bus(B1), risk_node(N).
%La acción load2 no ocurre si el número actual de habitantes no
%es 2. Recordemos que load2 evacúa a la segunda persona.
noaction_if(act(B1,load2(N)),neg(num_p_in_pob_1_2)):-bus(B1), risk_node(N).

%La acción load3 no ocurre si la posición del autobús B1 no es
%la población en riesgo N.
```

```
noaction_if(act(B1,load3(N)),neg(position(B1,N))) :- bus(B1), risk_node(N).
%La acción load3 no ocurre si el número actual de habitantes no
%es 1. Recordemos que load3 evacúa a la tercera persona.
noaction_if(act(B1,load3(N)),neg(num_p_in_pob_1_1)):-bus(B1), risk_node(N).

%La acción unload no ocurre si el autobús B1 no está en la población
%N la cual es una población refugio.
noaction_if(act(B1,unload(N)),neg(position(B1,N))) :- bus(B1),
                                                    refuge_node(N).

%Se agrega una nueva condición de ejecución para la acción evacuate.
%No ocurre la acción evacuate si el autobús no está ocupado.
noaction_if(act(B1,evacuate(I,F)),neg(status(B1,busy))):-bus(B1),
                                                    going_way(I,F).

%Se agrega una nueva condición de ejecución para la acción return.
%No ocurre la acción evacuate si el autobús no está vacío.
noaction_if(act(B1,return(I,F)),neg(status(B1,empty))):-bus(B1),
                                                    return_way(I,F).

%%CONSECUENCIAS DE EJECUCIÓN DE LAS ACCIONES.
%La primera consecuencia de la acción load1 es que ahora el número de
%habitantes en la población 1 es 2.
caused(num_p_in_pob_1_2,act(B1,load1(N))):- bus(B1), risk_node(N).
%La segunda consecuencia de la acción load1 es que ahora el número de
%habitantes en la población 1 ya no es 3.
```

```
caused(neg(num_hab_in_pob_1_3),act(B1,load1(N))):- bus(B1), risk_node(N).
%La tercera consecuencia de la acción load1 es que el número de
%habitantes en la población 1 no es 1.
caused(neg(num_p_in_pob_1_1),act(B1,load1(N))):- bus(B1), risk_node(N).
%La cuarta consecuencia de la acción load1 es que el número de
%habitantes en la población 1 no es 0.
caused(neg(num_p_in_pob_1_0),act(B1,load1(N))):- bus(B1), risk_node(N).
%La quinta consecuencia de la acción load1 es que el sentido del autobús
%B1 es hacia la zona de refugios en la población 1 no es 0.
caused(direction(B1,1),act(B1,load1(N))):- bus(B1), risk_node(N).
%La sexta consecuencia de la acción load1 es que el estado del autobús
%B1 está ocupado.
caused(status(B1,busy),act(B1,load1(N))):- bus(B1), risk_node(N).
%La septima consecuencia de la acción load1 es que el estado del autobús
%B1 ya no es vacío.
caused(status(B1,empty),act(B1,load1(N))):- bus(B1), risk_node(N).

%La primera consecuencia de la acción load2 es que ahora el número de
%habitantes en la población 1 es 1.
caused(num_p_in_pob_1_1,act(B1,load2(N))):- bus(B1), risk_node(N).
%La segunda consecuencia de la acción load2 es que ahora el número de
%habitantes en la población 1 ya no es 2.
caused(neg(num_p_in_pob_1_2),act(B1,load2(N))):- bus(B1), risk_node(N).
%La cuarta consecuencia de la acción load2 es que el sentido del autobús
%B1 es hacia la zona de refugios en la población 1 no es 0.
caused(direction(B1,1),act(B1,load2(N))):- bus(B1), risk_node(N).
```

```
%La quinta consecuencia de la acción load2 es que el estado del autobús
%B1 está ocupado.
caused(status(B1,busy),act(B1,load2(N))):- bus(B1), risk_node(N).
%La sexta consecuencia de la acción load2 es que el estado del autobús
%B1 ya no es vacío.
caused(neg(status(B1,empty)),act(B1,load2(N))):- bus(B1), risk_node(N).

%La primera consecuencia de la acción load3 es que ahora el número de
%habitantes en la población 1 es 0.
caused(num_p_in_pob_1_0,act(B1,load3(N))):- bus(B1), risk_node(N).
%La segunda consecuencia de la acción load3 es que ahora el número de
%habitantes en la población 1 ya no es 1.
caused(neg(num_p_in_pob_1_1),act(B1,load3(N))):- bus(B1), risk_node(N).
%La cuarta consecuencia de la acción load3 es que el sentido del autobús
%B1 es hacia la zona de refugios en la población 1 no es 0.
caused(direction(B1,1),act(B1,load3(N))):- bus(B1), risk_node(N).
%La quinta consecuencia de la acción load3 es que el estado del autobús
%B1 está ocupado.
caused(status(B1,busy),act(B1,load3(N))):- bus(B1), risk_node(N).
%La sexta consecuencia de la acción load3 es que el estado del autobús
%B1 ya no es vacío.
caused(neg(status(B1,empty)),act(B1,load3(N))):- bus(B1), risk_node(N).
```

Después de ejecutar el programa anterior se genera un solo plan el cual corresponde a un solo *answer set*. En la tabla 5.4 se presenta la secuencia de acciones, así como los cambios en el fluente que representa el número de habitantes y el fluente que representa

la posición.

TIEMPO	ACCIONES	FLUENTE `num_p`	FLUENTE `position`
1	load1(a1,1)	num_p_in_pob_1_3 neg(num_p_in_pob_1_2) neg(num_p_in_pob_1_1) neg(num_p_in_pob_1_0)	position(a1,1) neg(position(a1,3)) neg(position(a1,2))
2	evacuate(a1, 1, 2)	num_p_in_pob_1_2	position(a1,1)
3	evacuate(a1, 2, 3)	num_p_in_pob_1_2	position(a1,2)
4	unload(a1,3)	num_p_in_pob_1_2	position(a1,3)
5	return(a1,3,2)	num_p_in_pob_1_2	position(a1,3)
6	return(a1,2,1)	num_p_in_pob_1_2	position(a1,2)
7	load2(a1,1)	num_p_in_pob_1_1 neg(num_p_in_pob_1_3) neg(num_p_in_pob_1_2) neg(num_p_in_pob_1_0)	position(a1,1) neg(position(a1,3)) neg(position(a1,2))
8	evacuate(a1, 1, 2)	num_p_in_pob_1_1	position(a1,1)
9	evacuate(a1, 2, 3)	num_p_in_pob_1_1	position(a1,2)
10	unload(a1,3)	num_p_in_pob_1_1	position(a1,3)
11	return(a1,3,2)	num_p_in_pob_1_1	position(a1,3)
12	return(a1,2,1)	num_p_in_pob_1_1	position(a1,2)
13	load3(a1,1)	num_p_in_pob_1_0 neg(num_p_in_pob_1_3) neg(num_p_in_pob_1_2) neg(num_p_in_pob_1_1)	position(a1,1) neg(position(a1,3)) neg(position(a1,2))
14	evacuate(a1, 1, 2)	num_p_in_pob_1_0	position(a1,1)
15	evacuate(a1, 2, 3)	num_p_in_pob_1_0	position(a1,2)
16	unload(a1,3)	num_p_in_pob_1_0 neg(num_p_in_pob_1_3) neg(num_p_in_pob_1_2) neg(num_p_in_pob_1_1)	position(a1,3) neg(position(a1,1)) neg(position(a1,2))

Tabla 5.4: *Answer set* generado por SMODELS

Como podemos ver, en la secuencia de acciones ya no existe problema con los fluentes cada que ocurre la acción `load`. En cada tiempo sólo aparece como verdadero el fluente que representa el número actual de habitantes en la población 1 y se niegan el resto de fluentes. Con esto se consigue llegar a la meta con un plan que no se basa en información

falsa, ya que los fluentes que se están cumpliendo son los correctos.

5.4. Conclusiones

En este capítulo se amplió el dominio propuesto en el capítulo 4 y se modeló en SMODELS. Se consideró la capacidad en los caminos y en las poblaciones, el sentido de los caminos, el número de habitantes en las poblaciones en riesgo; modelamos la concurrencia entre acciones y la capacidad limitada en los autobuses para evacuar un número determinado de personas.

De manera general, SMODELS permitió modelar de manera fácil las características mencionadas anteriormente con lo cual podemos considerarlo como un sistema flexible y adecuado para el modelado de problemas de planificación, en este caso problemas de generación de planes de evacuación.

La manera en como SMODELS genera los planes, que corresponden a los *answer sets* es fácil de interpretar. Como ejemplo podemos ver las tablas 5.1, 5.2 y 5.4 en ellas se ve claramente como en cada instante de tiempo cada uno de los autobuses ejecuta las acciones, el tiempo en el cual cada autobús inicia su recorrido y el tiempo en el cual cada uno llega a la meta.

Un aspecto de SMODELS es que el número de habitantes en las poblaciones en riesgo no fue tan natural modelarlo, por lo que se propuso una solución la cual funcionó adecuadamente consiguiendo que el sistema llegará correctamente a la meta generando un plan de evacuación con información correcta. Aunque con esta solución si el número de poblaciones en riesgo y el número de habitantes en cada población son grandes el programa se extenderá mucho.

Otro aspecto de SMODELS es que conforme se trabaja con escenarios en los cuales el número de poblaciones y/o caminos se va haciendo más grande, el sistema se vuelve

más lento para generar los planes.

En el apéndice B se muestra otro ejemplo de un escenario que considera un número mayor de poblaciones y autobuses que los presentados en este capítulo. En este ejemplo no se modela el número de habitantes en las poblaciones en riesgo por las razones mencionadas anteriormente.

Capítulo 6

Ampliando el dominio del problema con DLVK

En este capítulo retomaremos el dominio modelado en el capítulo anterior el cual se define de acuerdo a las características que se presentaron en el capítulo 3 utilizando el enfoque DLVK, el cual es un *front-end* de DLV que sirve para modelar problemas de planificación. Modelaremos la capacidad en las poblaciones y en los caminos para recibir un número limitado de autobuses en un mismo instante de tiempo. Consideraremos el número de habitantes en las poblaciones ubicadas dentro de la zona de riesgo, los autobuses tendrán capacidad limitada para evacuar a las personas que se encuentran en la zona de riesgo además tendrán que ir y regresar de la zona de riesgo a los refugios, y viceversa, hasta que el número de habitantes en las poblaciones en riesgo sea igual a cero. Se tomará en cuenta el sentido de los caminos que en determinado momento puedan servir como ruta de evacuación. Se modelará la concurrencia entre acciones, en la cual las acciones ocurrieran de manera concurrente en un mismo instante de tiempo siempre y cuando sean realizadas por diferentes autobuses, en otras palabras, un mismo autobús no podrá realizar dos o más acciones diferentes al mismo tiempo

Todo lo anterior para conocer las características que nos brinda DLVK para modelar los problemas de planificación, principalmente los problemas de generación de planes

de evacuación.

6.1. Modelado del dominio en *AS Planning*

Como se mencionó anteriormente, el dominio que modelaremos en este capítulo se basa en el definido en el capítulo 5. Consideraremos el sentido en los caminos, la capacidad en poblaciones y caminos, así como el número de habitantes en las poblaciones que se ubican en la zona de riesgo. Para esto retomaremos los fluentes, las acciones, las condiciones de ejecución de las acciones, las consecuencias de ejecución de las acciones, las reglas de concurrencia, las reglas para modelar la capacidad en los caminos y poblaciones definidas en la sección 5.1.

Ejemplo 6.1. Retomando el ejemplo 5.1 se tiene una red la cual consta de 4 poblaciones. La población 1 y 4 se encuentran en la zona de riesgo. La población 3 es un refugio. Inicialmente existe un autobús en la población 1 y en la población 4, cada uno de estos autobuses tienen capacidad para evacuar a 100 personas en cada viaje. El número de habitantes en cada población en riesgo es de 200. La población 1 está concectada con la población 2 y esta última a la población 3. Por otra, parte la población 4 está conectada con la población 2. La población 1, 4 y 3 tienen capacidad para recibir a dos autobuses, mientras que la población 2 sólo puede recibir 1 autobús. Los caminos de 1 a 2 y de 4 a 2 tienen capacidad para dos autobuses, mientras que el camino de 2 a 3 tienen capacidad para 1. Todos los caminos pueden ser utilizados para ir hacia el refugio o para regresar a la zona de riesgos. La meta es que los autobuses *a1* y *a2* lleguen a la población refugio 3 y que el número de habitantes en las poblaciones en riesgo 1 y 4 sea 0.

El escenario descrito anteriormente se muestra en la figura 6.1

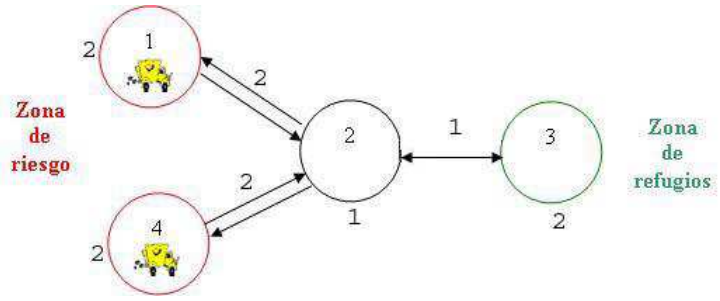

Figura 6.1: Red de poblaciones y caminos.

6.2. Modelado en DLVK

Recordemos que el modelado de un problema de planificación en DLVK se puede dividir en dos partes. En la primera parte se guarda el *background knowledge* del escenario en un archivo con extensión .bk. En la segunda parte se modelan los fluentes, las acciones, las condiciones de ejecución de las acciones, las consecuencias de ejecución de las acciones, las condiciones iniciales y la meta; todo esto en un archivo con extensión .plan. Esto nos permite que un mismo dominio pueda ser implementado en diferentes escenarios sin tener que comenzar desde cero. Simplemente se adecúa el *background knowledge* definido en el archivo .bk.

En esta sección presentamos la implementación del escenario mostrado en el ejemplo 6.1. pero ahora utilizando DLVK. La implementación retomará el modelado en DLVK que fue realizado en la sección 4.3 por lo que no se comenzará desde cero, simplemente bastará con agregar las nuevas características del dominio definidas en este capítulo.

Recordemos que la meta es que los autobuses `a1` y `a2` lleguen al refugio 3 y que además el número de habitantes en las poblaciones en riesgo 1 y 4 sea igual 0. A continuación se muestra el código en DLVK comenzando con la definición del *background knowledge*

en el archivo .bk.

```
%José Miguel Leyva Cruz UTM 28/09/2006
%ejemplo2.bk

%Se declara el rango máximo para los números enteros utilizados
%en el programa. Esta variable nos ayudará a modelar el número
%de habitantes.
#maxint=250.

%%%BACKGROUND KNOWLEDGE
%Conjunto de poblaciones. Donde cada población es un nodo.
node(1).  node(2).  node(3).  node(4).

%Lista de nodos considerados como refugios.
refuge_node(3).

%Lista de nodos en la zona de riesgo.
risk_node(1).
risk_node(4).

%Conjunto de segmentos.
%Conjunto de segmentos que permiten evacuar hacia la zona de refugios.
%going_way(población inicial, población final).
going_way(1,2).  going_way(4,2).  going_way(2,3).
```

```
%Conjunto de segmentos que permiten regresar a la zona de riesgo.
%return_way(población inicial, población final).
return_way(2,1).  return_way(2,4).  return_way(3,2).

%Dirección para indicar el sentido que llevan los autobuses.
%Si el valor es 1 se dirigen hacia la zona de los refugios.
%Si el valor es 0 se dirigen hacia la zona de riesgo.
%direction(valor).
direction(1).  direction(0).

%Se definen los autobuses.
%bus(identificador del autobús).
bus(a1).  bus(a2).

%Estado del autobús.
%Si el valor es full el autobús está lleno.
%Si el valor es empty el autobús está vacío.
status(empty).  status(full).

%Se declara la variable people de tipo entero para modelar el número
%de habitantes en las poblaciones en riesgo.
people(X):- #int(X), 0 <= X, X <= 200.
```

Una vez modelado el *background knowledge* se definen los nuevos fluentes, las acciones, las consecuencias de ejecución de las acciones, las condiciones de ejecución de las acciones, el estado inicial y la meta todo esto se almacena en un archivo con extensión

.plan. El siguiente código retoma el modelado que se hizo en DLVK en la sección 4.3.

```
%José Miguel Leyva Cruz UTM 28/09/2006
%ejemplo2.plan

%En este bloque se definen todos los fluentes.
fluents:
    %%Indica que la posición del autobús B1 es el nodo N.
    position(B1,N) requires bus(B1), node(N).

    %Indica el número de habitantes P en la población en riesgo N.
    num_p(N,P) requires risk_node(N), people(P).

    %Indica el sentido D del autobús B1.
    %Si D es igual a 1 el autobús se dirige hacia la zona de refugios.
    %Si D es igual a 1 el autobús se dirige hacia la zona de riesgo.
    way(B1,D) requires bus(A1), direction(D).

    %Indica el estado E del autobús B1.
    %Si E es full el autobús está lleno.
    %Si E es empty el autobús está vacío.
    statusbus(B1,E) requires bus(B1), status(E).

%En este bloque se definen todas las acciones.
actions:
    %evacuate: el autobús B1 viaja de la población inicial I a la
    %población final F, siempre que haya un camino de ida I-F.
```

```
    evacuate(B1, I, F) requires bus(B1), going_way(I,F).

    %load: carga al autobús B1 en la población de riesgo N.
    load(B1,N) requires bus(B1), risk_node(N).

    %unload: descarga el autobús B1 en el nodo refugio N.
    unload(B1,N) requires bus(B1), refuge_node(N).

    %return: el autobús B1 regresa a la zona de riesgo, en caso de
    %que sea necesario evacuar a más personas.
    return(B1, I, F) requires bus(B1), return_way(I,F).

%En este bloque se definen las condiciones y las consecuencias de
%ejecución para cada acción.
always:
%Condiciones de ejecución de las acciones.
    %La acción evacuate es posible si el autobús B1 está en la
    %población inicial I y además debe existir un camino que una a
    %población I con la población F.
    executable evacuate(B1, I, F) if position(B1,I), going_way(I,F).

    %La acción load es posible si el autobús se encuentra vacío
    %y además su posición es en un nodo refugio.
    executable load(B1,N) if statusbus(B1,empty), position(A1,N),
                             risk_node(N).
```

```
    %La acción unload ocurre si el autobús se encuentra lleno
    %y además su posición es en un nodo refugio.
    executable unload(B1,N) if statusbus(B1,full), position(B1,N),
                               refuge_node(N).

    %La acción return es posible si el autobús B1 está en la
    %población inicial I y además debe existir un camino de regreso
    %que una a la población I con la población F.
    executable return(B1, I, F) if position(B1,I), return_way(I,F).
    %La acción return es posible la dirección para el
    % autobús B1 es 0, es decir, el autobús B1 está
    %regresando a la zona de riesgo.
    executable return(B1, I, F) if way(B1, 0).

%Consecuencias de ejecución de las acciones.
    %La acción evacuate tiene las siguientes consecuencias:
    %La nueva posición del autobús B1 es la población final F.
    caused position(B1,F) after evacuate(B1,I,F).
    %La posición del autobús B1 ya no es la población inicial I.
    caused -position(B1,I) after evacuate(B1,I,F).

    %La acción load tiene las siguientes consecuencias:
    %El número de habitantes P_old en la población N disminuye.
    %P_new=P_old-100, donde el valor de 100 es la capacidad
    %que tiene el autobús para evacuar personas.
    caused num_p(N,P_new) if P_old=P_new+100 after load(B1,N), num_p(N,P_old).
```

```
%El número de habitantes en la poblacón N ya no es P_old.
caused -num_p(N,P_old) after load(A1,N), num_p(N,P_old).
%El sentido del autobús es hacia la zona de refugios.
caused way(B1,1) after load(A1,N).
%El sentido del autobús no es hacia la zona de riesgo.
caused -way(A1,0) after load(A1,N).
%El estado del autobús es lleno.
caused statusbus(A1,full) after load(A1,N).
%El estado del autobús ya no es vacío.
caused -statusbus(A1,empty) after load(A1,N).

%La acción unload tiene las siguientes consecuencias:
%El sentido del autobús es hacia la zona de riesgo.
caused way(A1,0) after unload(A1,N).
%El sentido del autobús no es hacia la zona de refugios.
caused -way(A1,1) after unload(A1,N).
%El autobús no está lleno.
caused -statusbus(A1,full) after unload(A1,N).
%El autobús está vacío.
caused statusbus(A1,empty) after unload(A1,N).

%La acción return tiene las siguientes consecuencias:
%La nueva posición del autobús B1 es la población final F.
caused position(A1,F) after return(A1,I,F).
%La posición del autobús B1 no es la población inicial I.
caused -position(A1,I) after return(A1,I,F).
```

```
%%%%RESTRICCIONES
    %Impide que el autobús a1 llegue a una población en riesgo que
    %no le corresponde evacuar. La palabra reservada forbidden se
    %utiliza en DLVK para restringir valores de los fluentes que uno
    %no quiere que ocurran. En este caso el autobús a1 no debe de llegar
    %a la población 4.
    forbidden position(a1,4).

%Se define la inercia en los fluentes. Esto es que los fluentes permanecen
%con sus valores hasta que no sean afectados directamente por la
%ejecución de una acción.
    inertial position(A1,N).
    inertial num_p(N,P).
    inertial way(A1,D).
    inertial statusbus(A1,E).

%%%CONCURRENCIA
    %La macro noCocurrency permite controlar la no concurrencia entre
    %acciones.
    noConcurrency.

%En este bloque se definen las condiciones iniciales.
initially:
    %La posición inicial del autobús a1 es la población 1.
    position(a1,1).
```

```
    %La posición inicial del autobús a2 es la población 4.
    position(a2,4).
    %El estado de los autobuses a1 y a2 inicialmente es vacío.
    statusbus(a1,empty).        statusbus(a2,empty).
    %Las poblaciones 1 y 4 inicialmente tiene 200 habitantes.
    num_p(1,200).       num_p(4,200).

%En este bloque se definen las condiciones finales o meta.
goal:
    %La posición final de los autobuses a1 y a2 es el refugio.
    position(a1,3),       position(a2,3),
    %El número final de habitantes en la población 1 es 0.
    num_p(1,0),
    %El número final de habitantes en la población 4 es 0.
    num_p(4,0) ? (19)
    %El parámetro que sigue después del signo de interrogación
    %indica el tiempo máximo posible que se requiere para alcanzar
    %la meta. En cada instante de tiempo se ejecuta una acción.
```

Para poder ejecutar el programa anterior se escribe la siguiente instrucción desde la línea de comandos.

```
DLV ejemplo2.plan ejemplo2.bk -FP
```

en la cual se está utilizando DLV para ejecutar el programa. Para activar el *front-end* DLVK se pone la bandera `-FP`.

Para nuestro ejemplo se generan varios planes, los cuales corresponden a los *answer*

sets. En la tabla 6.1 se presenta uno de ellos.

TIEMPO	ACCIONES
1	load(a2,4);
2	load(a1,1);
3	evacuate(a1,1,2);
4	evacuate(a1,2,3);
5	unload(a1,3);
6	return(a1,3,2);
7	return(a1,2,1);
8	load(a1,1);
9	evacuate(a1,1,2);
10	evacuate(a1,2,3);
11	evacuate(a2,4,2);
12	evacuate(a2,2,3);
13	unload(a2,3);
14	return(a2,3,2);
15	return(a2,2,4);
16	load(a2,4);
17	evacuate(a2,4,2);
18	evacuate(a2,2,3)

Tabla 6.1: *Answer set* generado por DLVK

6.3. Contribución

DLVK es un enfoque diseñado para modelar problemas de planificación por lo que el modelado de cada uno de los elementos de *AS Planning* se lleva a cabo de manera clara y sencilla. Esto se debe a que DLVK ya cuenta con una estructura en la cual cada elemento se define en secciones específicas lo cual hace al programa más fácil de entender, como se puede ver en la sección 4.3.

En DLVK fue muy fácil modelar los escenarios para las situaciones en riesgo tal y como se puede ver en la sección anterior con respecto a SMODELS. Uno de los aspectos que tiene DLVK es que se puede manejar de manera natural el número de habitantes

en las poblaciones. En DLVK se definió una operación en la cual el nuevo número de habitantes en la población en riesgo (`P_new`) fuera igual a la resta del número actual de habitantes en la población en riesgo (`P_actual`) menos el número de habitantes que está evacuando el autobús (`num_people`). Tal y como se ve en las siguientes líneas de código de DLVK:

```
caused num_p(N,P_new) if P_old=P_new+num_people after load(B1,N),num_p(N,P_old).
caused -num_p(N,P_old) after load(A1,N), num_p(N,P_old).
```

Esto permitió considerar que un autobús tiene que ir y regresar a la zona de riesgo tantas veces como sean necesarias hasta evacuar en su totalidad a los habitantes. Lo cual nos permite trabajar con escenarios más reales con respecto a [2].

Otro ventaja que nos brinda DLVK es que posee macros especiales que nos permiten modelar características tales como la no concurrencia entre acciones de forma rápida y sencilla. Para esto basta simplemente con definir dentro del programa la macro *no-Concurrency* y con esto las acciones ocurren de manera no concurrente. Esto hace que los programas en DLVK no sean tan extensos como en SMODELS, ya que al definir las macros dentro del programa, no es necesario escribir un conjunto de reglas que sirvan para modelar algunas características, como la no concurrencia. Algunas otras macros de DLVK pueden ser encontradas en [12].

Aunque DLVK permite modelar la concurrencia entre acciones ésta no es como se requiere para nuestro problema, ya que en los planes generados se quiere que las acciones ocurran de manera concurrente siempre y cuando cada una de ellas sea realizada por diferentes autobuses. En otras palabras, un mismo autobús no puede realizar más de una acción en un mismo instante de tiempo, lo cual por sentido común no es posible.

Retomando el ejemplo 6.1 un plan generado utlizando la macro `noConcurrency` es el

que se muestra en la tabla 6.2.

TIEMPO	ACCIONES
1	load(a2,4);
2	load(a1,1);
3	evacuate(a1,1,2);
4	evacuate(a1,2,3);
5	unload(a1,3);
6	return(a1,3,2);
7	return(a1,2,1);
8	load(a1,1);
9	evacuate(a2,4,2);
10	evacuate(a2,2,3);
11	unload(a2,3);
12	return(a2,3,2);
13	return(a2,2,4);
14	load(a2,4);
15	evacuate(a1,1,2);
16	evacuate(a2,4,2);
17	evacuate(a2,2,3);
18	evacuate(a1,2,3)

Tabla 6.2: *Answer set* generado por DLVK

Como se ve en la tabla anterior, el definir la macro `noConcurrency` hace que no se generen los planes como se requieren, ya que en un instante de tiempo sólo ocurre una acción realizada por un autobús y un diferente autobús debe esperar a que la acción anterior termine de ocurrir, además de que las acciones realizadas por un mismo autobús ocurren sin tener secuencia.

Para que las acciones sean concurrentes basta con no incluir la macro `noConcurrency` dentro del programa en DLVK, sin embargo esta concurrencia tampoco es como se requiere para modelar el problema. En este caso, la concurrencia genera planes en donde se ejecutan las acciones sin importar quién las está realizando.

Basándonos en el ejemplo 6.1. la tabla 6.3 muestra un plan sin la macro `noConcurrency`.

TIEMPO	ACCIONES
1	load(a1,1), load(a2,4);
2	evacuate(a1,2,3), evacuate(a2,2,3);
3	evacuate(a1,2,3), evacuate(a2,2,3), unload(a1,3), unload(a2,3);
4	return(a1,3,2), return(a2,3,2), load(a1,1), load(a2,4);
5	evacuate(a1,2,3), evacuate(a2,2,3), return(a1,2,1), return(a2,2,1);

Tabla 6.3: *Answer set* generado por DLVK

En el tiempo 3 del plan mostrado en la tabla 6.3 ocurren más de dos acciones diferentes realizadas por el autobús `a1`. Así también, ocurre lo mismo para el autobús `a2`, lo cual por sentido común no es posible. Como se ve esta opción tampoco genera planes como se requieren para el problema de modelado.

Como solución a lo anterior y para que los planes sean generados como el problema de modelado lo requiere, es decir, que en un mismo instante de tiempo puedan ocurrir varias acciones siempre y cuando éstas sean realizadas por diferentes autobuses, proponemos extender DLVK creando la macro `no_occurs_action`, la cual sería utilizada de la siguiente forma:

`no_occurs_action` *`accion1`* `if` *`accion2.`*

Para explicar mejor esta solución y su funcionamiento, retomemos las acciones del dominio modelado en el ejemplo 6.1. Se define la macro para cada una de las acciones de la siguiente manera:

```
no_occurs_action evacuate(B1,I,F) if load(B1,N).
no_occurs_action evacuate(B1,I,F) if unload(B1,N).
no_occurs_action evacuate(B1,I,F) if return(B1,I,F).
```

```
no_occurs_action return(B1,I,F) if load(B1,N).
no_occurs_action return(B1,I,F) if unload(B1,N).
no_occurs_action return(B1,I,F) if evacuate(B1,I,F).

no_occurs_action load(B1,N) if evacuate(B1,I,F).
no_occurs_action load(B1,N) if unload(B1,N).
no_occurs_action load(B1,N) if return(B1,I,F).

no_occurs_action unload(B1,N) if evacuate(B1,I,F).
no_occurs_action unload(B1,N) if load(B1,N).
no_occurs_action unload(B1,N) if return(B1,I,F).
```

Con lo anterior, se espera que DLVK permita la concurrencia tal y como se requiere para el modelado del problema, generando planes como se muestra en la tabla 6.4.

TIEMPO	ACCIONES	
1	load(a2,4);	load(a1,1);
2	evacuate(a2,4,2);	evacuate(a1,1,2);
3	evacuate(a2,2,3);	evacuate(a1,2,3);
4	unload(a2,3);	unload(a1,3)
5	return(a2,3,2);	return(a1,3,2);
6	return(a2,2,4);	return(a1,2,1);
7	load(a2,4);	load(a1,1);
8	evacuate(a2,4,2);	evacuate(a1,1,2);
9	evacuate(a2,2,3);	evacuate(a1,2,3);

Tabla 6.4: *Answer set* generado por DLVK utilizando la nueva macro `no_occurs_action`

Como se puede ver en cada instante de tiempo ocurren acciones diferentes o iguales pero realizadas por autobuses diferentes, con lo cual se consigue que los planes sean

generados tal y como nuestro problema lo requiere. Esto permitirá que se puedan ampliar los dominios en los cuales pueda ser utilizado DLVK.

Otra característica de DLVK es que no permite comparar acciones contra acciones antes de su ejecución ni durante su ejecución. Esto se detectó al querer modelar la capacidad de las poblaciones y de los caminos. Por ejemplo, si el autobús `B1` va a evacuar hacia la población `F` la cual tiene capacidad para un solo autobús, no podrá hacerlo si otro autobús `B2` está realizando también la acción evacuar hacia la misma población `F` y con esto la población ya no tendrá más capacidad para recibir a `B1`. Esto no se pudo modelar debido a que DLVK no permite comparar acciones contra acciones antes de que sean ejecutadas. Así también si en un tiempo `T` el autobús `B1` va a evacuar hacia la población `F` la cual está ocupada en su máxima capacidad podrá hacerlo sólo si en ese mismo tiempo `T` el autobús `B2`, que ya se encuentra en `F`, evacúa de `F` hacia otra población y con esto `F` puede recibir a `B1`. Esto tampoco se pudo modelar ya que DLVK no permite comparar las acciones durante su ejecución. Recordemos que este tipo de comparaciones son diferentes a las condiciones de ejecución ya que en éstas se verifica que los fluentes cumplan con las condiciones necesarias antes del comienzo de la ejecución de una acción.

En este trabajo de tesis se propone extender la sintaxis de DLVK para crear una nueva sección llamada `capacity` en donde se utilice la instrucción `no_executable` la cual nos servirá para definir las reglas que nos permitan modelar la capacidad de algunos elementos dependiendo del dominio del problema. En este caso la capacidad en los caminos y en las poblaciones. La nueva sección se define igual que la sección para los fluentes, las acciones, las condiciones iniciales, la meta, las condiciones de ejecución de las acciones y las consecuencias de ejecución de las acciones (véase el modelado en la sección 6.2.). Para explicar mejor la solución retomemos el ejemplo 6.1. Como primer paso se define en el archivo .bk los caminos y poblaciones con sus respectivas capacidades. Recorde-

mos que en el archivo .bk se modela el *background knowledge* de nuestro domino. El modelado queda de la siguiente manera:

```
%%Indica que la población N tiene capacidad C.
node_cap(1,2). node_cap(2,1). node_cap(3,2). node_cap(4,2).

%%Indica que el camino de I a F tiene capacidad C.
going_way(1,2,2). going_way(4,2,2). going_way(2,3,1).
return_way(2,1,2). return_way(2,4,2). return_way(3,2,1).
```

Posteriormente dentro del archivo .plan se crea la nueva sección llamada `capacity` en donde se definen las reglas necesarias para modelar la capacidad en caminos y poblaciones utilizando la instrucción `no_executable`. Recordemos también que el archivo .plan contiene los fluentes, las acciones, el estado inicial, el estado final o meta, las condiciones de ejecución de las acciones y las consecuencias de ejecución de las acciones. Las reglas dentro de la nueva sección quedan modeladas de la siguiente manera:

```
%En esta sección se definen reglas para modelar la capacidad en
%caminos y poblaciones.
capacity:
%%CAPACIDAD EN POBLACIONES%%
%El autobús B1 no podrá evacuar hacia la población F si ésta tiene
%capacidad para un autobús y el autobús B2 también está evacuando
%hacia la misma población F. Además B1 y B2 son diferentes.
    no_executable evacuate(B1,I,F) if node_cap(F,1), evacuate(B2,I,F),
                                      B2<>B1.
```

```
%El autobús B1 no podrá regresar a la población F si ésta tiene
%capacidad para un autobús y el autobús B2 también está regresando
%a la misma población F. Además B1 y B2 son diferentes.
    no_executable return(B1,I,F) if node_cap(F,1), return(B2,I,F),
                                    B2<>B1.

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 ya está en la población F.
%Además B1 y B2 son diferentes.
    no_executable evacuate(B1,I,F) if node_cap(F,1), position(B2,F),
                                    B2<>B1.

%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 ya está en la población F.
%Además B1 y B2 son diferentes.
    no_executable return(B1,I,F) if node_cap(F,1), position(B2,F),
                                    B2<>B1.

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 también están
%evacuando hacia la misma población F. Además B1, B2 y B3 son
%diferentes.
    no_executable evacuate(B1,I,F) if node_cap(F,2), evacuate(B2,I,F),
                                    evacuate(B3,I,F), B2<>B1, B3<>B1.
```

```
%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 también están
%regresando hacia la misma población F. Además B1, B2 y B3 son
% diferentes.
    no_executable return(B1,I,F) if node_cap(F,2), return(B2,I,F),
                                    return(B3,I,F), B2<>B1, B3<>B1.

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 ya están en la
%población F. Además B1, B2 y B3 son diferentes.
    no_executable evacuate(B1,I,F) if node_cap(F,2), position(B2,F),
                                    position(B3,F), B2<>B1, B3<>B1.

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 ya están en la
%población F. Además B1, B2 y B3 son diferentes.
    no_executable return(B1,I,F) if node_cap(F,2), position(B2,F),
                                    position(B3,F), B2<>B1, B3<>B1.

%%CAPACIDAD EN CAMINOS%%
%El autobús B1 no podrá evacuar por el camino de I a F si éste tiene
%capacidad para un sólo autobús y el autobús B2 está evacuando
%por el mismo camino de I a F. Además B1 y B2 son diferentes.
    no_executable evacuate(B1,I,F) if going_way(I,F,1),
                                    evacuate(B2,I,F), B2<>B1.
```

```
%El autobús B1 no podrá regresar por el camino de I a F si éste tiene
%capacidad para un sólo autobús y el autobús B2 está regresando
%por el mismo camino de I a F. Además B1 y B2 son diferentes.
    no_executable return(B1,I,F) if return_way(I,F,1),
                                    return(B2,I,F), B2<>B1.

%El autobús B1 no podrá evacuar por el camino de I a F si éste tiene
%capacidad para dos autobuses y los autobuses B2 y B3 están evacuando
%por el mismo camino de I a F. Además B1, B2 y B3 son diferentes.
    no_executable evacuate(B1,I,F) if going_way(I,F,2), evacuate(B2,I,F),
                                    evacuate(B3,I,F), B2<>B1, B2<>B3.

%El autobús B1 no podrá regresar por el camino de I a F si éste tiene
%capacidad para dos autobuses y los autobuses B2 y B3 están regresando
%por el mismo camino de I a F. Además B1, B2 y B3 son diferentes.
    no_executable return(B1,I,F) if return_way(I,F,2), return(B2,I,F),
                                    return(B3,I,F), B2<>B1, B2<>B3.
```

Cabe mencionar que DLVK cuenta con la instrucción `nonexecutable` la cual puede servir para excluir acciones que se están ejecutando simultáneamente y es utilizada como sigue:

```
nonexecutable A if B
```

La regla anterior indica que la ejecución de la acción `A` está prohibida si se cumple `B`. Al querer implementar está regla para modelar las capacidades en los caminos y en las poblaciones no se generaron los resultados requeridos, ya que nos interesa que dicha

regla sea utilizada de la siguiente manera:

`nonexecutable return(B1,I,F) if` **node_cap(I,F,1),** `return(B2,I,F), B2<>B1.`

La instrucción `nonexecutable` no respeta la condición `node_cap(F,1)` en la cual se indica que la población `F` tiene capacidad para recibir un autobús. Con nuestra propuesta se pretende que dicha condición sea tomada en cuenta ya que es importante saber la capacidad en las poblaciones y en los caminos para aplicar así la regla adecuada que prohiba que otro autobús pueda viajar a una población si ésta se encuentra a su máxima capacidad.

6.4. Conclusiones

Como se puedes ver el hecho de que DLVK sea una herramienta dirigida a modelar problemas de planificación nos permite utilizarla en los problemas de generación de planes de evacuación de una manera clara y sencilla. DLVK permite modelar cada uno de los elementos de *AS Planning* en secciones diferentes, esto es, se crea una sección en la cual se definen los fluentes, otra donde se definen las acciones, otra donde se definen las consecuencias de ejecución de las acciones y las condiciones de ejecución de las acciones y otras dos secciones donde se define las condiciones iniciales y la meta respectivamente. Esto hace que el programa sea más claro y pueda ser entendido de manera más rápida que en SMODELS.

Con DLVK se pudo modelar de manera sencilla el número de habitantes en las poblaciones en riesgo, cosa que recordemos en SMODELS no fue tan natural, por lo que se tuvo que proponer una segunda solución.

El hecho de que DLVK sea dirigido totalmente a modelar problemas de planificación ocasiona que no sea tan flexible como SMODELS. Lo anterior no nos permitió modelar algunas características tales como las capacidades en las poblaciones y en los caminos.

Como solución se propueso extender DLVK creando dentro de su estructura una nueva sección llamada `capacity` en la cual se definen las reglas para modelar la capacidad en las poblaciones y caminos.

Otro aspecto de DLVK es que no permite generar los planes como el modelado del problema lo requiere, esto es que exista concurrencia entre acciones siempre y cuando éstas sean ejecutadas por diferentes autobuses. Como solución a esto se propuso extender DLVK creando la macro `no_occurs_action` con lo cual las acciones ocurrirían de manera concurrente siempre y cuando no sean realizadas por el mismo autobús.

En el apéndice C mostramos otro ejemplo de un escenario implementado en DLVK en donde se consideran más poblaciones y autobuses que los presentados en este capítulo.

Capítulo 7

Conclusiones

En esta tesis se ha retomado el trabajo realizado en [2] en el cual se modelaron problemas de generación de planes de evacuación. Recordemos que en dicho trabajo se modelan con *Answer Set Programming* (ASP) algunos escenarios inspirados en la situación de riesgo del volcán Popocatépetl para obtener los planes de evacuación. Sin embargo, estos escenarios sólo consideran unos cuantos elementos: poblaciones en riesgo, poblaciones donde se encuentran los refugios, autobuses que son usados para evacuar a la población, caminos en un solo sentido y que conectan a las poblaciones. Además los caminos y los autobuses tienen una capacidad ilimitada. De manera general el trabajo realizado en [2] presenta los fundamentos teóricos de que ASP puede ser utilizado para modelar problemas de generación de planes de evacuación para la situación de riesgo del volcán Popocatépetl.

En este trabajo de tesis hemos extendido el uso de ASP, utilizando principalmente el enfoque *Answer Set Planning (AS Planning)*, en la generación de planes de evacuación para situaciones de riesgo de manera general y no enfocándonos solamente al caso del volcán Popocatépetl. Llevamos a la práctica el modelado de escenarios más apegados al mundo real con respecto a los ejemplos modelados en [2] tomando en cuenta elementos como la capacidad de refugios y caminos, el número de habitantes en los lugares de ries-

go, el sentido de los caminos, etc., para así finalmente generar los planes de evacuación. Se analizó qué tan flexibles son los sistemas SMODELS y DLV, en este caso utilizando principalmente el front-end DLVK, para llevar a la práctica el modelado de problemas de generación de planes de evacuación en ASP.

Se trabajó con *AS Planning* con el objetivo de conocer cómo se lleva a cabo el modelado de nuestro dominio utilizando los elementos de dicho enfoque: fluentes, acciones, condiciones de ejecución de las acciones, consecuencias de ejecución de las acciones, estado inicial y estado final o meta; usando para esto los lenguajes de acción, en este caso se utilizó el lenguaje de acción A. Se implementó cada uno de los elementos de *AS Planning* en SMODELS y DLVK utilizando un pequeño escenario para conocer la forma en cómo se realiza el modelado en cada uno de estos sistemas. Posteriormente se implementó en cada sistema el dominio definido para nuestro problema en base a lo propuesto en [11] y [5]. Finalmente, se propone extender DLVK y SMODELS para permitir la implementación de algunos elementos de nuestro dominio cuyo modelado no fue tan natural de realizar en cada sistema.

De manera general, coincidimos con [2] en que ASP puede ser utilizado para modelar problemas de generación de planes de evacuación para situaciones en riesgo. Se pudo comprobar en base en nuestra experiencia al trabajar con ASP que el agregar o modificar elementos del dominio consiste simplemente en agregar o eliminar una o dos líneas más al modelo y no volver a programar el código que nos ayude a encontrar el plan buscado. Todo esto debido a que ASP al ser declarativo permite que una misma definición de acciones, fluentes, estados inciales, consecuencias de ejecución de las acciones, metas y condiciones de ejecución de las acciones pueden ser utilizados para diferentes situaciones de riesgo.

Basándonos en los resultados obtenidos al implementar los escenarios, SMODELS permitió modelar de manera natural las capacidades en las poblaciones y caminos, lo cual

en DLVK no fue posible realizar. Por lo anterior, fue necesario proponer una solución que nos permita modelar dicha característica en DLVK, la cual consistió en extender el enfoque creando una nueva sección en la que se definan las capacidades de las poblaciones y caminos.

La concurrencia entre acciones se maneja de manera distinta en cada uno de los sistema. SMODELS permite manejar la concurrencia como ocurre en un caso real, esto es, permite que ocurran varias acciones en un instante de tiempo siempre y cuando dichas acciones sean realizadas por diferentes autobuses, lo que significa también, que un mismo autobús no puede realizar dos o más acciones en un mismo instante de tiempo. Esto hace que los planes sean generados como se requiere en nuestro problema de modelado. Pero en DLVK no ocurre esto, ya que la manera en como se maneja la concurrencia genera planes en donde un mismo autobús está realizando dos o más acciones en un mismo instante de tiempo, lo cual por sentido común sabemos que esto no es lo indicado para generar planes de evacuación. Debido a lo anterior se propuso crear en DLVK una nueva macro que nos permita generar los planes como se mencionó anteriormente.

En DLVK se pudo modelar de manera natural el número de habitantes en las poblaciones en riesgo implementado una regla en la cual el nuevo número de habitantes en la población en riesgo fuera igual al número actual de habitantes en la población en riesgo menos el número de habitantes que se estaban evacuando. Pero esto, no se pudo modelar en SMODELS de manera natural ya que al implementar la regla anterior el sistema generaba planes con información incorrecta, por lo que en SMODELS se propuso una segunda solución en la cual se definió un fluente por cada habitante en cada población en riesgo y varias acciones de acuerdo al número de habitantes que existen en la población en riesgo para que cada que se evacúe a un habitante, las consecuencias de ejecución de cada acción hagan que se cumplan como falsos aquellos fluentes que no correspondan al número actual de habitantes en un determinado instante y únicamente

se cumpla como verdadero el fluente que corresponda al número actual de habitantes. Con esto se consiguió que el sistema llegará a la meta generando un plan de evacuación con información correcta. Pero con esta solución, si el número de poblaciones en riesgo y el número de habitantes en cada población son grandes, el programa se extenderá mucho y ya no será tan claro.

Otro aspecto de SMODELS, es que por ser un sistema de propósito general, conforme se trabaja con escenarios en los cuales el número de poblaciones y/o caminos se va haciendo más grande, el sistema se vuelve más lento para generar los planes, lo cual no ocurre en DLVK ya que por ser un front-end diseñado para modelar problemas de planificación los planes se generan más rápidamente por el sistema. Esto último también nos permitió utilizar DLVK de una manera clara y sencilla ya que cuenta con secciones diferentes para modelar cada uno de los elementos de *AS Planning*, esto es, se crea una sección en la cual se definen los fluentes, otra donde se definen las acciones, otra donde se definen las consecuencias de ejecución de las acciones y las condiciones de ejecución de las acciones, y otras dos secciones donde se define el estado final y el estado inicial o meta respectivamente. Esto hace que el programa sea más claro y pueda ser entendido de manera más rápida que en SMODELS. Además DLVK cuenta con macros especiales para modelar características como la no concurrencia entre acciones con lo cual al definir dicha macro no es necesario escribir un conjunto de reglas que ocasionan que el programa se vuelva tan extenso como ocurre en SMODELS.

El resumen de este trabajo de tesis fue presentado en el Cuarto Congreso Nacional de Ciencias de la Computación [3] organizado por la Facultad de Ciencias de la Computación de la Benemérita Universidad Autónoma de Puebla, llevándose a cabo en las instalaciones de dicha universidad.

7.1. Trabajo a futuro

Finalmente, como trabajo a futuro proponemos extender el enfoque DLVK de tal manera que permita modelar problemas de planes de evacuación de manera natural y que cuente con las características que SMODELS tiene para modelar el problema de planes de evacuación. Implementar en DLVK la macro `no_occurs_action`, definida en la sección 6.3, para manejar la concurrencia entre acciones, la cual permita que las acciones ocurran siempre de manera concurrente cuando éstas sean realizadas por diferentes autobuses. Crear una nueva sección llamada `capacity` en la cual se pueda utilizar la instrucción `no_executable` para que se pueda modelar el problema de las capacidades en los caminos y poblaciones, como se ve en la sección 6.3.
También se propone ampliar el dominio modelado con el objetivo de trabajar con escenarios más apegados al mundo real con respecto a [2] y a este trabajo de tesis, entre algunos de los elementos que proponemos son los siguientes:

- Considerar la capacidad de los refugios para recibir un número limitado de personas.
- Considerar como escenarios mapas geográficos reales publicados, por ejemplo, por instituciones gubernamentales.
- Crear mapas de rutas de evacuación de acuerdo a los planes generados por los sistemas.
- Modelar la posición de los vehículos no sólo considerando las poblaciones en las que se encuentran, sino también en el camino por el cual se encuentran viajando.
- Para cada población asignarle prioridad a cada uno de los caminos que llevan a él. Por ejemplo, si varios vehículos quieren llegar a una misma población en el

mismo instante de tiempo y por diferentes caminos, llegarán a ella de acuerdo a la prioridad asignada al camino por el cual van a viajar.

- Considerar niveles de riesgo en la zona de peligro.

Basándonos en que ASP es un lenguaje que sirve para modelar problemas de generación de planes de evacuación de acuerdo a los resultados obtenidos en [2] y en esta tesis, se propone utilizar alguna herramienta de desarrollo y junto con los sistemas DLVK o SMODELS para crear una aplicación gráfica que permita a futuros usuarios modelar escenarios de situaciones de riesgo para generar planes de evacuación de una manera clara y sencilla.

Apéndice A

Enfoques para problemas de planificación

Como se mencionó anteriormente ASP no es el único enfoque que permite modelar problemas de planificación. En [6] se proponen otros enfoques, los cuales se presentan a continuación de manera resumida:

- **Planificación basada en lógica.** También llamado planeación basada en cambios. Está basado en:

  - α**:** designa un estado inicial.
  - Γ**:** designa un conjunto de acciones.
  - ρ**:** designa un objetivo.
  - Ω**:** es un conjunto de condiciones acerca del estado inicial.

  La ejecución es de la siguiente manera: el *planner* trata de generar un plan Γ el cual es ejecutado por el *modulo de acción* o *ejecutor*. Cuando el sistema se encuentra en un estado i y dicho estado cumple con las condiciones iniciales, resultarán nuevos estados hasta hallar el estado g que satisfaga la condiciones del estado objetivo. Este enfoque también ayuda con la solución que existe en los proble-

mas de planificación: *"Frame Problem", "Qualification Problem" y "Ramification Problem"* [9].

- **Planificación basada en operadores.** Todas las *acciones* son representadas por *operadores*. Este enfoque, utiliza varios *esquemas de operadores* o *representación de planes*. Utilizando este enfoque se pueden representar:

  - **Diseños de esquemas de operador:** listas de agregar-eliminar, representación procedural vs. representación declarativa (NONLIN vs. NOAH), etc.
  - **Diseños de representación de planes:** planes lineales, planes no lineales, planes jerárquicos (abstractos), planes de orden parcial, planes condicionales, etc.
  - **Algoritmos de planeación:** planeación como búsqueda, planeación de orden parcial, progresión, regresión, etc.
  - **Planes críticos:** reformulación de planes, reparación, orden total, etc.

- **Algoritmos de planificación.** Introduce el concepto de *planeación como búsqueda.* Hay dos enfoques:

  - **Buscar un estado del mundo.** Cada nodo en el grafo denota un estado del mundo. Los arcos en el grafo corresponden a la ejecución de una acción específica. El problema de planeación consiste en encontrar una ruta que lleve de un estado inicial a un estado objetivo. Hay dos algoritmos:
    - *Progresión:* este algoritmo realiza la búsqueda de un estado objetivo a través de nuevos estados generados por las acciones que son realizadas en cada estado, inicia a partir del estado inicial

- *Regresión:* este algoritmo realiza la búsqueda hacia atrás partiendo del estado objetivo buscando acciones cuyos efectos satisfagan uno o más de los objetivos planteados, y fijando las condiciones previas de las acciones elegidas como objetivos.

Ambos algoritmos son completos y seguros. En la mayoría de las situaciones la regresión es la mejor estrategia.

- **Buscar un estado del plan.** Cada nodo en el grafo representa un *plan parcial*. Los arcos representan *operaciones que modifican los planes*. Se puede buscar un plan con una secuencia ordenada de acciones (planeación de orden total), o un plan con un conjunto de acciones ordenadas parcialmente (planeación de orden parcial (POP)).

  La planeación de orden parcial tiene tres componentes:

  - *Un conjunto de acciones.*
  - *Un conjunto de condicionales.*
  - *Un conjunto de consecuencias.*

- **Planificación basada en casos.** Dado un nuevo problema, un objetivo y una descripción del estado inicial se busca en la librería de casos un caso similar al que se está presentando que cumpla con la descripción de los estados iniciales y los objetivos. Se modifica la solución recuperada de acuerdo al nuevo problema y se almacena nuevamente en la librería para casos futuros.

Apéndice B

Ampliando el escenario en SMODELS

En este apéndice presentamos un escenario más grande que el definido en el capítulo 5 con la finalidad de implementarlo en SMODELS. La implementación utiliza como base los fluentes, las acciones, las condiciones y las consecuencias de ejecución de las acciones definidos en la sección 5.3. Básicamente, las modificaciones consisten en adecuar el *background knowledge* al nuevo escenario, recordemos que esto es una de las grandes ventajas que tiene ASP.

B.1. Definición del escenario

Se cuenta con una red que consta de 12 poblaciones. Existe una zona de riesgo en la cual se ubican las poblaciones 1, 2 y 3 las cuales cuentan con un autobús para evacuar a los habitantes. Existe una zona de refugios en la cual se ubican las poblaciones 10, 11 y 12. El resto de las poblaciones sirven como tránsito entre la zona de riesgo y los refugios. Las poblaciones tienen capacidades diferentes para recibir autobuses: las poblaciones 1, 2, 3, 5, 8, 10, 11 y 12 tienen capacidad para recibir a un autobús en un instante de tiempo, por lo que si otro autobús desea pasar por alguna de ellas deberá esperar a que

ésta sea desocupado. Las poblaciones 4, 6, 7 y 9 tienen capacidad para dos autobuses por lo que si un tercer autobús desea pasar por alguna de ellas deberá esperar a que sea desocupada por alguno de los autobuses que ya están en dichas poblaciones. Los caminos pueden ser en un sólo sentido o en ambos, es decir, pueden ser utilizados tanto para salir de la zona de riesgo como para regresar a ella. Al igual que las poblaciones algunos caminos tienen capacidades diferentes: los caminos de 1 a 4, de 4 a 5, de 7 a 10 y de 5 a 6 tienen capacidad para un autobús, por lo que si un segundo autobús desea viajar por este camino debe de esperar sea desocupado. Finalmente los caminos de 2 a 5, de 3 a 6, de 6 a 8, de 5 a 7, de 5 a 8, de 8 a 9, de 7 a 11, de 9 a 11, de 9 a 12 y de 4 a 2 tienen capacidad para dos autobuses. El escenario descrito anteriormente se muestra en la figura B.1.

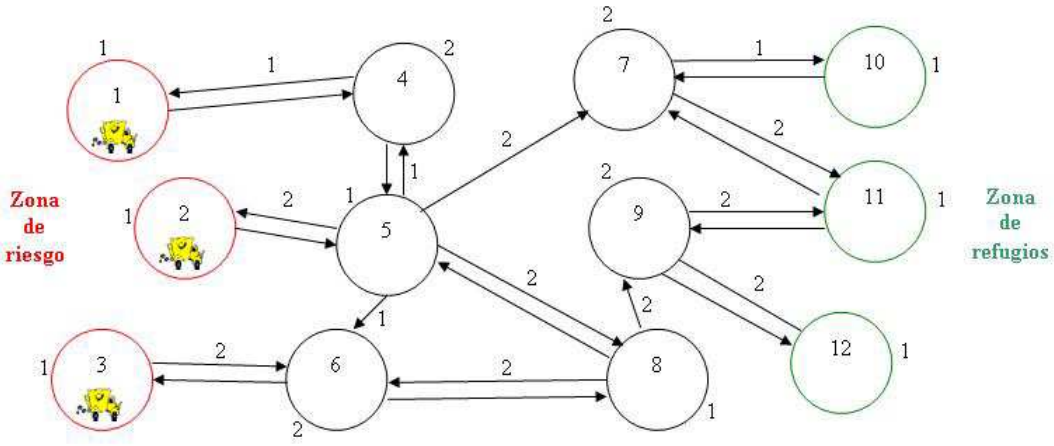

Figura B.1: Red de poblaciones y caminos.

A continuación se modela el escenario utilizando SMODELS en base a la figura B.1 y los elementos definidios en la sección 5.2.

```
%José Miguel Leyva Cruz, 20/11/2006 UTM
```

```
%ejemplo3.sm

%Se declara una constante númerica n la cual representa el tiempo
%máximo posible que se requiere para alcanzar la meta. En cada
%instante de tiempo se ejecuta una acción.
const n=13.
time(1..n).
% BACKGROUND KNOWLEDGE
% Conjunto de poblaciones. Donde cada población es un nodo.
%node_cap(población, capacidad).
node_cap(1,1).  node_cap(2,1).  node_cap(3,1).  node_cap(4,2).
node_cap(6,2).  node_cap(7,2).  node_cap(8,1).  node_cap(9,2).
node_cap(11,1).  node_cap(12,1).  node_cap(5,1).  node_cap(10,1).

%Lista de nodos considerados como refugios.
refuge_node(10).  refuge_node(11).  refuge_node(12).

%Conjunto de caminos y capacidad de cada camino.
%Conjunto de caminos que permiten evacuar hacia la zona de refugios.
%going_way(población inicial, población final, capacidad).
going_way(1,4,1).  going_way(2,5,2).  going_way(3,6,2).  going_way(6,8,2).
going_way(4,5,1).  going_way(5,7,2).  going_way(5,8,2).  going_way(8,9,2).
going_way(7,10,1). going_way(7,11,2). going_way(9,11,2). going_way(9,12,2).

%Conjunto de segmentos que permiten regresar a la zona de riesgo.
%return_way(población inicial, población final, capacidad)
```

```
return_way(4,1,1). return_way(4,2,2). return_way(5,2,2). return_way(5,6,1).
return_way(6,3,2). return_way(8,6,2). return_way(5,4,1). return_way(8,5,2).
return_way(9,8,2). return_way(10,7,1). return_way(11,7,2). return_way(11,9,2).
return_way(12,9,2).

%Dirección para indicar el sentido que llevan los autobuses.
%Si el valor es 1 se dirigen hacia la zona de refugios.
%Si el valor es 0 se dirigen hacia la zona de riesgo.
%direction(valor).
direction(0). direction(1).

%Se definen los autobuses.
%bus(identificador del autobús).
bus(a1). bus(a2). bus(a3).

%Condiciones iniciales. Posición inicial y sentido de los autobuses.
%position(bus, ubicación)
%sentido(bus, dirección)
initially(position(a1,1)).    initially(way(a1,1)).
initially(position(a2,2)).    initially(way(a2,1)).
initially(position(a3,3)).    initially(way(a3,1)).

%Objetivos finales o metas. Posición final y sentido de los autobuses.
finally(position(a1,1)).    finally(way(a1,0)).
finally(position(a2,2)).    finally(way(a2,0)).
finally(position(a3,3)).    finally(way(a3,0)).
```

```
%Se verifica que la meta sea alcanzada.
%No se quiere que no haya un plan, es decir, se quiere que haya un plan.
:- not plan.

%Existe un plan si se llega a la meta a lo más en n pasos.
plan :- goal(n).

%La meta se cumple en el tiempo T si no hay evidencia de que la
%meta no haya sido cumplida.
goal(T) :- time(T),
       not not_goal(T).

%La meta no se cumple en el tiempo T si no hay evidencia de que se han
%cumplido las condiciones finales (finally).
not_goal(T) :- time(T),
       literal(X),
       finally(X),
       not hold(X,T).

%%%FLUENTES
%Indica que la posición del autobús B1 es el nodo N.
fluent(position(B1,N)) :- bus(B1), node_cap(N,C).

%Indica el valor de la dirección D del autobús B1.
%Si D es igual a 1 se dirige hacia la zona de refugios.
```

```
%si D es igual a 0 se dirige hacia la zona en riesgo.
fluent(way(B1,D)):-bus(B1), direction(D).

%%%ACCIONES
%evacuate: el autobús B1 viaja de la población inicial I a la
%población final F siempre que haya un camino de ida de I-F.
action(act(B1,evacuate(I,F))) :- going_way(I,F,C), bus(B1).

%return: el autobús B1 viaja de la población inicial I a la
%población final F siempre que haya un camino de regreso de I-F.
action(act(B1,return(I,F))) :- return_way(I,F,C), bus(B1).

%turn: el autobús B1 gira en el nodo refugio N para regresar a la
%zona de riesgo.
action(act(B1,turn(N))):- bus(B1), refuge_node(N).

%%%CONSECUENCIAS DE EJECUCIÓN DE LAS ACCIONES
%La primera consecuencia de la acción evacuate es que la nueva posición
%del autobús B1 es la población final F.
caused(position(B1,F),act(B1,evacuate(I,F))):-
                                    bus(B1), going_way(I,F,C).
%La segunda consecuencia de la acción evacuate es que la posición
%del autobús B1 ya no es la población inicial I.
caused(neg(position(B1,I)),act(B1,evacuate(I,F))) :-
                                    bus(B1), going_way(I,F,C).
```

```
%La primera consecuencia de la acción return es que la nueva posición
%del autobús B1 es F.
caused(position(B1,F),act(B1,return(I,F))):-
                                        bus(B1),
                                        return_way(I,F,C).
%La segunda consecuencia de la acción return es que la posición del
%autobús B1 ya no es la población inicial I.
caused(neg(position(B1,I)),act(B1,return(I,F))):-
                                        bus(B1), return_way(I,F,C).

%La primera consecuencia de la acción turn es que el sentido del
%autobús B1 cambia a 0, es decir, el autobús se encuentra en dirección
%hacia la zona en riesgo.
caused(way(B1,0),act(B1,turn(N))):- bus(B1), refuge_node(N).
%La segunda consecuencia de la acción turn es que el sentido del
%autobús B1 ya no es 1, es decir, el autobús no se encuentra en dirección
%hacia la zona de refugios.
caused(neg(way(B1,1)),act(B1,turn(N))):- bus(B1), refuge_node(N).

%%%CONDICIONES DE EJECUCIÓN DE LAS ACCIONES
%La acción evacuate no es posible si el autobús B1 no está en la
%población inicial I y además debe existir un camino que una a la
%población I con la población F.
noaction_if(act(B1,evacuate(I,F)),neg(position(B1,I))):-
                                                bus(B1), going_way(I,F,C).
%La acción evacuate no es posible si el sentido del autobús
```

```
%B1 no es 1, es decir, hacia la la zona de refugios.
noaction_if(act(B1,evacuate(I,F)),neg(way(B1,1))):-
                                                bus(B1), going_way(I,F,C).

%La acción return no es posible si el autobús B1 no está en la
%población inicial I y además debe existir un camino de regreso que una
%a la población I con la población F.
noaction_if(act(B1,return(I,F)),neg(position(B1,I))):-
                                                bus(B1), return_way(I,F,C).
%La acción evacuate no es posible si el sentido del autobús
%B1 no es 1, es decir, hacia la zona de refugios.
noaction_if(act(B1,return(I,F)),neg(way(B1,0))):-
                                                bus(B1), return_way(I,F,C).

%La acción turn no es posible si el autobús B1 no se encuentra en
%un nodo refugio N.
noaction_if(act(B1,turn(N)),neg(position(B1,N))):-bus(B1), refuge_node(N).

%%%RESTRICCIONES DE EJECUCIÓN
%La acción A no es ejecutable en el tiempo T si las condiciones de
%ejecución para A no se cumplen.
not_executable(A,T) :-
            action(A),
            time(T),
            noaction_if(A,F),
            hold(F,T).
```

```
%La acción A es ejecutable en el tiempo T si no hay evidencia
%de que A no pueda ser ejecutada.
executable(A,T) :-
       action (A),
       time (T),
       not not_executable(A,T).

% Es posible ejecutar la acción A en el tiempo T si no hay evidencia
%de que la meta no haya sido cumplida.
possible(A,T) :-
       action(A),
       executable(A,T),
       time(T),
       not goal(T).

%El valor del fluente F cambia su valor en el tiempo T+1 si la
%acción A afecta a F. Además la acción A es ejecutable y está
%ocurriendo en el tiempo T.
hold(F,T+1) :- literal(F),
       time(T),
       T<n,
       action(A),
       executable(A,T),
       occurs(A,T),
       caused(F,A).
```

```
%El fluente F siempre se cumple en el tiempo 1 se define
%como condición inicial. Esto se cumple para todos los fluentes
%definidos como initially.
hold(F,1) :- literal(F), initially(F).

%El fluente F no se cumple en el tiempo 1 si no está definido
%como condición inicial. Esto se cumple para todos los fluentes
%que no están definidos como initially.
hold(neg(F), 1) :- fluent(F), not hold(F,1).

%Si F es un fluente, su valor contrario es -F.
%Recordemos que la negación de un fluente se escribe con el símbolo -.
contrary(F, neg(F)) :- fluent(F).

%Si -F es un fluente, su valor contrario es F.
contrary(neg(F), F) :- fluent(F).

%La literal G es un fluente.
literal(G) :- fluent(G).

%La negación de la literal G también es un fluente.
literal(neg(G)) :- fluent(G).

%La literal F mantiene su valor en el tiempo T+1 si no hay
%evidencia de que cambie a su valor contrario G. Esto se conoce como
```

```
%la ley de la inercia, un fluente mantiene su valor hasta que no haya
%evidencia de que éste haya cambiado.
hold(F, T+1) :-literal(F),
       literal(G),
       contrary(F,G),
       time(T),
       T<n,
       hold(F,T),
       not hold(G, T+1).

% La acción A ocurre en el tiempo T si no hay evidencia de que la acción
%no pueda ocurrir en dicho instante.
occurs(A,T) :-
       action(A),
       time(T),
       possible(A,T),
       not not_occurs(A,T).

%La acción A ocurre en el tiempo T si se verifica que A es posible y no
%existe evidencia de que no pueda ocurrir en dicho tiempo.
occurs(A,T) :-
       action(A),
       time(T),
       possible(A,T),
       verify(A,T).
```

```
%Se definen subacciones internas. Esto se utilizará más
%adelante para el manejo de la concurrencia.
sub_action(evacuate(I,B1)) :- going_way(I,B1,C).
sub_action(return(I,B1)) :- return_way(I,B1,C).
sub_action(turn(N)):- refuge_node(N).

%%%CONCURRENCIA
%El autobús B1 no puede realizar la acción A1 en el tiempo T si ya está
%realizando otra acción A2. Además A1 y A2 son diferentes.
not_occurs(act(B1,A1),T) :-
       action(act(B1,A1)), sub_action(A1),bus(B1),
       action(act(B1,A2)), sub_action(A2),
       time(T),
       occurs(act(B1,A2),T),
       neq(A1,A2).

%%%CAPACIDAD EN CAMINOS
%El autobús B1 no podrá evacuar por el camino I-F si éste tiene
%capacidad para un sólo autobús y el autobús B2 está evacuando
%en el mismo tiempo T por el camino I-F. Además B1 y B2 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
       time(T),
       bus(B1),
       going_way(I,F,1),
       occurs(act(B2,evacuate(I,F)),T), bus(B2),
       neq(B1,B2).
```

```
%El autobús B1 no podrá regresar por el camino I-F si éste tiene
%capacidad para un sólo autobús y el autobús B2 está regresando
%en el mismo tiempo T por el camino I-F. Además B1 y B2 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
       time(T),
       bus(B1),
       return_way(I,F,1),
       occurs(act(B2,return(I,F)),T), bus(B2),
       neq(B1,B2).

%El autobús B1 no podrá evacuar por el camino I-F si éste tiene
%capacidad para dos autobuses y los autobuses B2 y B3 están evacuando
%en el mismo tiempo T por el camino I-F. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
       time(T),
       bus(B1),
       going_way(I,F,2),
       occurs(act(B2,evacuate(I,F)),T), bus(B2),
       occurs(act(B3,evacuate(I,F)),T), bus(B3),
       neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá regresar por el camino I-F si éste tiene
%capacidad para dos autobuses y los autobuses B2 y B3 están regresando
%en el mismo tiempo T por el camino I-F. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
```

```
        time(T),
        bus(B1),
        return_way(I,F,2),
        occurs(act(B2,return(I,F)),T), bus(B2),
        occurs(act(B3,return(I,F)),T), bus(B3),
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%%%CAPACIDAD EN LAS POBLACIONES
%El autobús B1 no podrá evacuar hacia la población F si ésta tiene
%capacidad para un autobús y el autobús B2 también está evacuando
%hacia la misma población F en el mismo tiempo T. Además B1 y B2
%son diferentes.
not_occurs(act(B1,evacuate(I1,F)),T):-
        time(T),
        bus(B1),
        node_cap(I1,C1),
        node_cap(F,1),
        occurs(act(B2,evacuate(I2,F)),T), bus(B2), node_cap(I2,C2),
        neq(B1,B2).

%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 también está regresando
%hacia la misma población F en el mismo tiempo T. Además B1 y B2
%son diferentes.
not_occurs(act(B1,return(I1,F)),T):-
        time(T),
```

```
        bus(B1),
        node_cap(I1,C1),
        node_cap(F,1),
        occurs(act(B2,return(I2,F)),T), bus(B2), node_cap(I2,C2),
        neq(B1,B2).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 ya está en la población F
%en el mismo tiempo T. Además B1 y B2 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C1),
        hold(position(B2,F),T),
        node_cap(F,1),
        bus(B2),
        neq(B1,B2).

%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para un autobús y el autobús B2 ya está en la población F
%en el mismo tiempo T. Además B1 y B2 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C1),
        hold(position(B2,F),T),
```

```
        node_cap(F,1),
        bus(B2),
        neq(B1,B2).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 también están
%evacuando hacia la misma población F en el mismo tiempo T. Además
%B1, B2 y B3 son diferentes.
not_occurs(act(B1,evacuate(I1,F)),T):-
        time(T),
        bus(B1),
        node_cap(I1,C),
        node_cap(F,2),
        occurs(act(B2,evacuate(I2,F)),T), bus(B2), node_cap(I2,C),
        occurs(act(B3,evacuate(I3,F)),T), bus(B3), node_cap(I3,C),
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá regresar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 también están
%regresando hacia la misma población F en el mismo tiempo T. Además
% B1, B2 y B3 son diferentes.
not_occurs(act(B1,return(I1,F)),T):-
        time(T),
        bus(B1),
        node_cap(I1,C),
        node_cap(F,2),
```

```
        occurs(act(B2,return(I2,F)),T), bus(B2), node_cap(I2,C),
        occurs(act(B3,return(I3,F)),T), bus(B3), node_cap(I3,C),
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 ya están en la
%población F en el mismo tiempo T. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,evacuate(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C),
        hold(position(B2,F),T),
        hold(position(B3,F),T),
        node_cap(F,2),
        bus(B2), bus(B3),
        neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 no podrá evacuar por la población F si ésta tiene
%capacidad para dos autobuses y los autobuses B2 y B3 ya están en la
%población F en el mismo tiempo T. Además B1, B2 y B3 son diferentes.
not_occurs(act(B1,return(I,F)),T):-
        time(T),
        bus(B1),
        node_cap(I,C),
        hold(position(B2,F),T),
        hold(position(B3,F),T),
```

```
       node_cap(F,2),
       bus(B2), bus(B3),
       neq(B1,B2), neq(B1,B3), neq(B2,B3).

%El autobús B1 podrá evacuar hacia la población F, la cual se
%encuentra ocupada en su máxima capacidad, si se verifica que en
%el mismo tiempo T algún autobús que ya se encuentra en la población
%F evacua hacia otra población.
verify(act(B1,evacuate(I1,F)),T):-
       time(T),
       bus(B1),
       going_way(I1,F,C1),
       node_cap(F,X),
       occurs(act(B2,evacuate(F,F2)),T), bus(B2),
       going_way(F,F2,C2),
       neq(B1,B2).

%El autobús B1 podrá regresar hacia la población F, la cual se
%encuentra ocupada en su máxima capacidad, si se verifica que en
%el mismo tiempo T algún autobús que ya se encuentra en la población
%F regresa hacia otra población.
verify(act(B1,return(I1,F)),T):-
       time(T),
       bus(B1),
       return_way(I1,F,c1),
       node_cap(F,X),
```

```
occurs(act(B2,return(F,F2)),T), bus(B2),
return_way(F,F2,c2),
neq(B1,B2).
```

Para poder ejecutar el programa se escribe la siguiente instrucción desde la línea de comandos.

lparse < ejemplo3.sm | smodels 0

La tabla B.1 presenta la secuencia de acciones de un plan generado por el sistema, el cual corresponde a un *answer set*.

TIEMPO	ACCIONES		
1	evacuate(a1, 1, 4)	evacuate(a2, 2, 5)	evacuate(a3, 3, 6)
2	no_action	evacuate(a2, 5, 7)	evacuate(a3, 6, 8)
3	evacuate(a1, 4, 5)	evacuate(a2, 7, 11)	evacuate(a3, 8, 9)
4	evacuate(a1, 5, 8)	turn(a2, 11)	evacuate(a3, 9, 12)
5	evacuate(a1, 8, 9)	return(a2, 11, 9)	turn(a3, 12)
6	evacuate(a1, 9, 11)	return(a2, 9, 8)	turn(a3, 12)
7	turn(a1,11)	return(a2, 8, 5)	turn(a3, 12)
8	return(a1, 11, 9)	return(a2, 5, 4)	return(a3, 12, 9)
9	return(a1, 9, 8)	return(a2, 4, 2)	return(a3, 9, 8)
10	return(a1, 8, 5)	no_action	return(a3, 8, 6)
11	return(a1, 5, 4)	no_action	return(a3, 6, 3)
12	return(a1, 4, 1)	no_action	no_action

Tabla B.1: *Answer set* generado por SMODELS

Apéndice C

Ampliando el escenario en DLVK

En este apéndice presentamos un escenario más grande que el definido en el capítulo 6 con la finalidad de implementarlo en DLVK. La implementación utliza como base los fluentes, las acciones, las condiciones y las consecuencias de ejecución de las acciones definidos en la sección 6.2. Básicamente las modificaciones consiste en adecuar el *background knowledge* al nuevo escenario, recordemos que esto es una de las grandes ventajas que tiene ASP.

C.1. Definición del escenario

Se cuenta con una red que consta de 5 poblaciones. Existe una zona de riesgo en la cual se ubican las poblaciones 1 y 2 con 200 habitantes cada una. En cada población en la zona de riesgo se encuentra un autobús con capacidad para evacuar a 100 personas por cada viaje. También existe una zona de refugios en la cual se ubican las poblaciones 4 y 5. El resto de las poblaciones sirven como tránsito entre la zona de riesgo y la zona de refugios. La meta es que el número de habitantes en las poblaciones en riesgo sea igual a 0 y que la posición final de los autobuses sea un refugio. Los caminos son en ambos sentidos, tanto puede servir para ir a la zona de refugios como para regresar

hacia la zona de riesgo. Recordemos que en DLVK no es posible modelar la capacidad en las poblaciones y en los caminos, por lo que supondremos que los autobuses pueden transitar sin ningún problema. El escenario descrito anteriormente se muestra en la figura C.1.

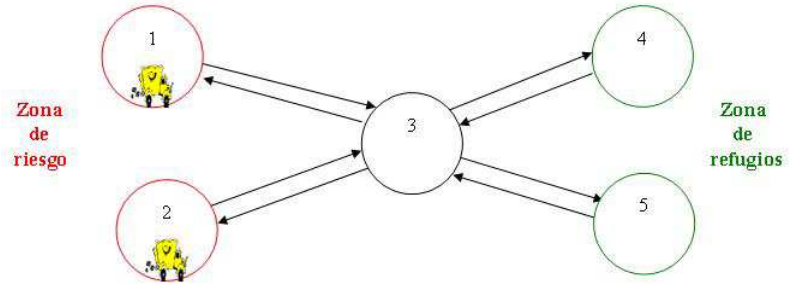

Figura C.1: Red de poblaciones y caminos

A continuación se modela el escenario utilizando DLVK en base a la figura C.1 y los elementos definidos en la sección 5.3. Para definir el *background knowledge* en DLVK se crea un archivo con extensión .bk en el cual se define el nuevo escenario.

```
%José Miguel Leyva Cruz UTM 30/09/2006
%ejemplo3.bk

%Se declara el rango máximo para los números enteros utilizados
%en el programa. Esta variable nos ayudará a modelar el número
%de habitantes.
#maxint=250.
%%%BACKGROUND KNOWLEDGE

%Conjunto de poblaciones. Donde cada población es un nodo.
```

```
node(1).  node(2).  node(3).  node(4).  node(5).

%Lista de nodos considerados como refugios.
refuge_node(5).  refuge_node(4).

%Lista de nodos en la zona de riesgo.
risk_node(1).  risk_node(2).

%Conjunto de segmentos.
%Conjunto de segmentos que permiten evacuar hacia la zona de refugios.
%going_way(población inicial, población final).
going_way(1,3).  going_way(2,3).  going_way(3,5).  going_way(3,4).

%Conjunto de segmentos que permiten regresar a la zona de riesgo.
%return_way(población inicial, población final).
return_way(3,1).  return_way(5,3).  return_way(4,3).  return_way(3,2).

%Dirección para indicar el sentido que llevan los autobuses.
%Si el valor es 1 se dirigen hacia la zona de los refugios.
%Si el valor es 0 se dirigen hacia la zona de riesgo.
%direction(valor).
direction(1).
direction(0).

%Se definen los autobuses.
%bus(identificador del autobús).
```

```
bus(a1).
bus(a2).

%Estado del autobús.
%Si el valor es full el autobús está lleno.
%Si el valor es empty el autobús está vacío.
status(empty).
status(full).

%Se declara la variable people de tipo entero para modelar el número
%de habitantes en las poblaciones en riesgo.
people(X):- #int(X), 0 <= X, X <= 200.
```

Los fluentes, las acciones, las consecuencias y las condiciones de ejecución de las acciones, el estado inicial y el estado final o meta se definen en un archivo con extensión .plan, basándonos en lo que se tiene en la sección 5.3.

```
%José Miguel Leyva Cruz UTM 30/09/2006
%ejemplo3.plan

%En este bloque se definen todos los fluentes.
fluents:
    %%Indica que la posición del autobús B1 es el nodo N.
    position(B1,N) requires bus(B1), node(N).

    %Indica el número de habitantes P en la población en riesgo N.
```

```
    num_p(N,P) requires risk_node(N), people(P).

    %Indica el sentido D del autobús B1.
    %Si D es igual a 1 el autobús se dirige hacia la zona de refugios.
    %Si D es igual a 1 el autobús se dirige hacia la zona de riesgo.
    way(B1,D) requires bus(B1), direction(D).

    %Indica el estado E del autobús B1.
    %Si E es full el autobús está lleno.
    %Si E es empty el autobús está vacío.
    statusbus(B1,E) requires bus(B1), status(E).

%En este bloque se definen todas las acciones.
actions:
    %evacuate: el autobús B1 viaja de la población inicial I a la
    %población final F, siempre que haya un camino de ida I-F.
    evacuate(B1, I, F) requires bus(B1), going_way(I,F).

    %load: carga al autobús B1 en la población de riesgo N.
    load(B1,N) requires bus(B1), risk_node(N).

    %unload: descarga el autobús B1 en el nodo refugio N.
    unload(B1,N) requires bus(B1), refuge_node(N).

    %return: el autobús B1 regresa a la zona de riesgo, en caso de
    %que sea necesario evacuar a más personas.
```

```
    return(B1, I, F) requires bus(B1), return_way(I,F).

%En este bloque se definen las condiciones y las consecuencias de
%ejecución para cada acción.
always:
%Condiciones de ejecución de las acciones.
    %La acción evacuate es posible si el autobús B1 está en la
    %población inicial I y además debe existir un camino que una a
    %población I con la población F.
    executable evacuate(B1, I, F) if position(B1,I), going_way(I,F).

    %La acción load es posible si el autobús se encuentra vacío
    %y además su posición es en un nodo riesgo.
    executable load(B1,N) if statusbus(B1,empty), position(A1,N),
                             risk_node(N).

    %La acción unload ocurre si el autobús se encuentra lleno
    %y además su posición es en un nodo refugio.
    executable unload(B1,N) if statusbus(B1,full), position(B1,N),
                               refuge_node(N).

    %La acción return es posible si el autobús B1 está en la
    %población inicial I y además debe existir un camino de regreso
    %que una a la población I con la población F.
    executable return(B1, I, F) if position(B1,I), return_way(I,F).
    %La acción return es posible la dirección para el
```

```
    % autobús B1 es 0, es decir, el autobús B1 está
    %regresando a la zona de riesgo.
    executable return(B1, I, F) if way(B1, 0).

%Consecuencias de ejecución de las acciones.
    %La acción evacuate tiene las siguientes consecuencias:
    %La nueva posición del autobús B1 es la población final F.
    caused position(B1,F) after evacuate(B1,I,F).
    %La posición del autobús B1 ya no es la población inicial I.
    caused -position(B1,I) after evacuate(B1,I,F).

    %La acción load tiene las siguientes consecuencias:
    %El número de habitantes P_old en la población N disminuye.
    %P_new=P_old-100, donde el valor 100 es la capacidad
    %que tiene el autobús para evacuar personas.
    caused num_p(N,P_new) if P_old=P_new+100 after load(B1,N), num_p(N,P_old).
    %El número de habitantes en la poblacón N ya no es P_old.
    caused -num_p(N,P_old) after load(A1,N), num_p(N,P_old).
    %El sentido del autobús es hacia la zona de refugios.
    caused way(B1,1) after load(A1,N).
    %El sentido del autobús no es hacia la zona de riesgo.
    caused -way(A1,0) after load(A1,N).
    %El estado del autobús es lleno.
    caused statusbus(A1,full) after load(A1,N).
    %El estado del autobús ya no es vacío.
    caused -statusbus(A1,empty) after load(A1,N).
```

```
    %La acción unload tiene las siguientes consecuencias:
    %El sentido del autobús es hacia la zona de riesgo.
    caused way(A1,0) after unload(A1,N).
    %El sentido del autobús no es hacia la zona de refugios.
    caused -way(A1,1) after unload(A1,N).
    %El autobús no está lleno.
    caused -statusbus(A1,full) after unload(A1,N).
    %El autobús está vacío.
    caused statusbus(A1,empty) after unload(A1,N).

    %La acción return tiene las siguientes consecuencias:
    %La nueva posición del autobús B1 es la población final F.
    caused position(A1,F) after return(A1,I,F).
    %La posición del autobús B1 no es la población inicial I.
    caused -position(A1,I) after return(A1,I,F).

%%%%RESTRICCIONES
    %Impide que el autobús a1 llegue a una población en riesgo que
    %no le corresponde evacuar. La palabra reservada forbidden se
    %utiliza en DLVK para restringir valores de los fluentes que uno
    %no quiere que ocurran.
    forbidden position(a1,2).

    %Impide que el autobús a1 llegue a un refugio que no le corresponde.
    %Lo anterior en caso de que el refugio sea ya uno determinado por
```

```
    %las características del problema.
    forbidden position(a1,4).

%Se define la inercia en los fluentes. Esto es que los fluentes permanecen
%con sus valores hasta que no sean afectados directamente por la
%ejecución de una acción.
    inertial position(A1,N).
    inertial num_p(N,P).
    inertial way(A1,D).
    inertial statusbus(A1,E).

%%%CONCURRENCIA
    %La macro noCocurrency permite controlar la no concurrencia entre
    %acciones.
    noConcurrency.

%En este bloque se definen las condiciones iniciales.
initially:
    %La posición inicial del autobús a1 es la población 1.
    position(a1,1).
    %La posición inicial del autbús a2 es la población 2.
    position(a2,2).
    %El estado de los autobuses a1 y a2 inicialmente es vacío.
    statusbus(a1,empty).
    statusbus(a2,empty).
    %Las poblaciones 1 y 2 inicialmente tiene 200 habitantes.
```

```
    num_p(1,200).
    num_p(2,200).

%En este bloque se definen las condiciones finales o meta.
goal:
    %La posición final del autobús a1 es la población
    %5 la cual es un refugio.
    position(a1,5),
    %La posición final del autobús a2 es la población
    %4 la cual es un refugio.
    position(a2,4),
    %El número final de habitantes en la población 1 es 0.
    num_p(1,0),
    %El número final de habitantes en la población 2 0.
    num_p(2,0) ? (19)
    %El parámetro que sigue después del signo de interrogación
    %indica el tiempo máximo posible que se requiere para alcanzar
    %la meta. En cada instante de tiempo se ejecuta una acción.
```

Para poder ejecutar el programa se escribe la siguiente instrucción desde la línea de comandos.

DLV ejemplo3.plan ejemplo3.bk -FP

en la cual se está utilizando DLV para ejecutar el programa. Para activar el front-end DLVK se pone la bandera `-FP`.

Para nuestro ejemplo se generan varios planes, los cuales corresponden a los *answer*

sets. En la tabla C.1 presentamos únicamente un plan.

TIEMPO	ACCIONES
1	load(a2,2)
2	load(a1,1)
3	evacuate(a1,1,3)
4	evacuate(a1,3,5)
5	unload(a1,5)
6	return(a1,5,3)
7	return(a1,3,1)
8	load(a1,1)
9	evacuate(a1,1,3)
10	evacuate(a1,3,5)
11	unload(a1,5)
12	evacuate(a2,2,3)
13	evacuate(a2,3,5)
14	unload(a2,5)
15	return(a2,5,3)
16	return(a2,3,2)
17	load(a2,2)
18	evacuate(a2,2,3)
19	evacuate(a2,3,4)

Tabla C.1: *"Answer set"* generado por DLVK

Bibliografía

[1] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

[2] Claudia Zepeda Cortés. *Evacuation Planning using Answer Sets.* PhD thesis, Universidad de las Americas-Puebla and Institut National des Sciencies Appliqus de Lyon, 2005.

[3] José Miguel Leyva Cruz; Claudia Zepeda Cortés. Representación de planes de evacuación usando answer set programming. In *To appear in the memories of the Cuarto Congreso Nacional de Ciencias de la Computación (BUAP).*

[4] Nicola Leone; Gerald Pfeifer; Wolfgang Faber. *The DLV Project. A Disjunctive Datalog System (and more).* [Online]: http://www.dbai.tuwien.ac.at/proj/dlv/ (Última consulta: 21/03/2007).

[5] S.A. Tajandra; H.W. Hamacher. Mathematical modelling of evacuation problems: A state of art. *Institut Techno-und Wirtsdchaftsmathmatik*, 24, 2001.

[6] Deepak Kumar. *AI Planning.* [Online]: http://mainline.brynmawr.edu/d̃kumar/UGAI/planning.html#references (Última consulta: 06/09/2006).

[7] Michael Gelfond; Vladimir Lifschitz. *The Stable Model Semantics for Logic Programming. 5th Conference on Logic Programming.* Mit Press, 1988.

[8] Michael Gelfond; Vladimir Lifschitz. Action lenguajes. *Electronic Transactions on Artificial Intelligence (EATI)*, 2:193–210, 1998.

[9] J. McCarthy. *Concepts of logical Artificial Intelligence*. Stanford University, 2000. [Online]: http://www-formal.stanford.edu/jmc/concepts-ai.html (Última consulta: 02/02/2007).

[10] Erick T. Mueller. *Commonsense Reasoning*. Elsevier.

[11] Office of the United Nations Disaster Relief Coordinator (UNDRO) and United Nations Educational Scientific Cultural Organization (UNESCO). *Volcanic Emergency Management*. United Nations, New York, 1985. [Online]: http://volcanoes.usgs.gov/About/What/Reduce/DevelopPlans.html (Última consulta: 12/07/2006).

[12] Thomas Eiter; Wolfgang Faber; Nicola Leone; Gerald Pfeifer; Axel Polleres. Planning under incomplete knowledge. In *Computational Logic*, pages 807–821, 2000.

[13] I.Ñiemel; P. Simons. Smodels-an implementation of the stable model and well-founded semantics for normal logic programs. *Lecture Notes in Artificial Intelligence*, 1265:420–429, 1997.