Chapter 1

Resumen en Español

1.1 Introducción

Actualmente, los responsables de la protección contra situaciones de desastre deben tomar decisiones sobre la preparación y ejecución de los planes de evacuación considerando causas potenciales de desastre. Por lo tanto, sería deseable desarrollar un sistema capaz de obtener y de analizar los planes de la evacuación. Donde este sistema este basado en el conocimiento sobre su ambiente particular, los datos geográficos y sus propias capacidades. Además, debe ser capaz de intercambiar información y servicios con sistemas similares así como con las personas. Un formalismo que proponemos para desarrollar tal sistema es Answer Set Programming (ASP). ASP es un lenguaje de programación lógico y declarativo para la representación del conocimiento [14]. ASP representa un nuevo paradigma para la programación lógica que permite, manejar problemas con conocimiento por defecto y razonamiento no-monotónico mediante con el concepto de la negación como falla.

Específicamente, el objetivo de nuestro trabajo es *investigar y evaluar las capacidades de ASP para representar situaciones del desastre con el objetivo de dar ayuda en la definición de planes de evacuación.* La motivación de nuestro trabajo se baso

en la idea que ASP cuanta con muchas de las capacidades que tal sistema debe tener: Es posible traducir la información geográfica a un formato que ASP pueda manejar. Existe *Answer Set Planning* que proporciona una manera natural y elegante de modelar los problemas de planificación [16]. ASP utiliza el concepto de negación como falla que permite expresar excepciones, restricciones y representar conocimiento incompleto. Además en ASP existen diversos enfoques para expresar preferencias.

Para lograr nuestro objetivo, comenzamos estudiando el formato de la información geográfica. Basados en nuestra propia experiencia proponemos un procedimiento para extraer el conocimiento necesario para representar la zona del peligro a partir de la información geográfica. Además consideramos que normalmente en una zona en riesgo hay un conjunto de rutas de evacuación predefinidas. Sin embargo, en un caso real es posible que parte de estas rutas predefinidas se bloqueen. En este caso la definición de planes de evacuación alternativos es necesaria. Utilizamos *Reglas para restaurar consistencia* (reglas-RC) para obtener los planes de evacuación alternativos. También mostramos que los programas con reglas-RC se pueden representar correctamente usando programas con disyunción ordenada (ODLP).

Utilizamos el Lenguaje $\mathcal{PP}$ para expresar preferencias a diversos niveles sobre los planes alternativos. Definimos el lenguaje $\mathcal{PP}^{par}$, como extensión del lenguaje $\mathcal{PP}$ donde sus conectivos permiten que representemos de forma compacta las preferencias que tienen una característica en particular. Además, en este trabajo presentamos una breve descripción sobre la relación entre la lenguaje $\mathcal{PP}$ y la lógica lineal proposicional temporal (LTL), puesto que consideramos que el lenguaje $\mathcal{PP}$ podría tomar ventaja del marco de trabajo de LTL para expresar preferencias.

También desarrollamos una extensión de ODLP a una clase más amplia de programas lógicos. Es importante mencionar que un subconjunto de estos programas son útiles para permitir una codificación más simple y más fácil de obtener los planes

preferidos con respecto a las preferencias expresadas en $\mathcal{PP}$. Finalmente, introducimos la noción de *Contenido Semántico de un programa* como un punto de vista alternativo para obtener variantes de la semántica de Answer Sets. Una de estas variantes es una nueva semántica que introducimos en este trabajo, llamada *answer sets parciales*.

1.2 Antecedentes

1.2.1 Answer Set Programming

El uso de Answer Set Programming (ASP) permite describir un problema computacional como un programa lógico cuyos answer sets corresponden a las soluciones de problema dado. En este trabajo, *un programa es interpretado como una teoría proposicional* y la única negación usada es la *negación como falla*. Para lectores no familiarizados con este enfoque, recomendamos [38, 33] como lectura posterior. Por lo tanto nuestra discusión se limita a programas proposicionales.

Usamos *el lenguaje de la lógica proposicional* de la manera usual, donde los símbolos proposicionales son $p, q, \ldots$; los conectivos proposicionales son $\wedge, \vee, \rightarrow, \bot$, y los símbolos auxiliares son $(,)$. Una *fórmula proposicional bien formada* es definida inductivamente como sigue: Un símbolo proposicional es una fórmula proposicional bien formada. Si f y g son fórmulas proposicionales bien formadas, entonces también lo son $\neg f$, $f \wedge g$, $f \vee g$, $f \rightarrow g$. Una fórmula proposicional bien formada puede ser llamada sólo *fórmula*. Un *átomo* es un símbolo proposicional. Dada cualquier fórmula bien formada f, asumimos que $\neg f$ es exactamente la abreviación de $f \rightarrow \bot$ y $\top$ es la abreviación de $\bot \rightarrow \bot$. Hacemos énfasis en que la única negación usada en este trabajo es la *negación como falla* y es representada por el símbolo $\neg$. Vale la pena mencionar que siempre es posible manejar la otra negación llamada *clásica* o también *negación fuerte*, denotada por

$-$, por medio de transformar los átomos con negación clásica [15]. Cada átomo con negación clásica, $-a$, que ocurre en la formula es sustituido por un nuevo átomo, a°, y la regla $\neg(a \wedge a^\circ)$ es agregada. La regla $\neg(a \wedge a^\circ)$ puede también ser escrita como $(a \wedge a^\circ) \rightarrow \bot$ basada en el hecho de que $\neg f$ es solo una abreviación de $f \rightarrow \bot$.

La *firma de un programa* P, denotada como $\mathcal{L}_P$, es el conjunto de átomos que ocurren en P. Una *literal* es cualquier átomo p (una literal positiva) o la negación de un átomo $\neg p$ (una literal negativa). Una *literal negada* es el signo de negación $\neg$ seguido por una literal, i.e. $\neg p$ or $\neg\neg p$. En particular, $f \rightarrow \bot$ es llamada *restricción* y también es denotada como $\leftarrow f$. Una *teoría regular* o *programa lógico* es solo un conjunto de formulas bien formadas, y puede ser llamado solamente *teoría* o *programa* cuando no surjan ambigüedades.

En algunas definiciones usamos *lógica intuicionista*, la cual será denotada por I. Para un conjunto dado de átomos M y un programa P, escribiremos $P \vdash_{\mathrm{I}} M$ para abreviar [27] para toda $a \in M$. Para un conjunto dado de átomos M y un programa P, escribiremos $P \Vdash_{\mathrm{I}} M$ para denotar que: $P \vdash_{\mathrm{I}} M$ y que P es consistente con respecto a I (es decir, que no hay una formula A tal que $P \vdash_{\mathrm{I}} A$ y $P \vdash_{\mathrm{I}} \neg A$). Algunas veces escribiremos $\vdash$ en lugar de $\vdash_{\mathrm{I}}$ cuando no surjan ambigüedades. Ahora definimos los answer sets (o modelos estables) de programas lógicos. La semántica de answer sets fue primero definida en términos de la así llamada *reducción Gelfond-Lifschitz* [14] donde es estudiada en el contexto de transformaciones sobre el programa dependientes de la sintaxis. Nosotros seguimos un enfoque alternativo estudiado por Pearce [38] y también estudiado por Osorio et.al. [33]. Este enfoque caracteriza los answer sets de una teoría proposicional en términos de lógica intuicionista y es presentada en el siguiente teorema. La notación está basada en [33].

Theorem 1.1. *Sea P una teoría y M un conjunto de átomos. M es un answer set*

para P *sii* $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$.

Como notación adicional que será utilizada en el resto de este trabajo tenemos lo siguiente: Un conjunto finito de formulas P es *consistente* [27] si no hay alguna formula A tal que A y $\neg A$ sean teoremas de P. Un conjunto finito de formulas P es *completo* [27] si, para cualquier formula A de P, $P \vdash A$ o $P \vdash \neg A$. Un conjunto finito de formulas P' es una *extensión* de un conjunto finito de formulas P [27] si cada teorema de P es un teorema de P'. Dados dos conjuntos X y Y, escribimos $X \subset Y$ para denotar que $X \subseteq Y$ and $X \neq Y$.

Como de costumbre en Answer Set Programming, consideramos que un programa con símbolos del predicado es solamente una abreviatura del programa sin variables, denotado como $sinVariables(P)$.

En algunos casos, necesitamos modelar algunos problemas usando símbolos del predicado con variables, donde estas variables deben ser instanciadas solamente con un subconjunto del universo de Herbrand del programa. Esta clase de programas se llama *programas de clases con símbolos de predicado* [4]. Por ejemplo, utilizaremos programas de clases con símbolos de predicado cuando estudiamos la codificación de un problema de planificación en la sección 1.2.4.

Actualmente, hay varios resolvedores para Answer Sets. En este trabajo hemos usado DLV[1] , SMODELS[2] , un frontend para DLV llamado DLVK[3] , una modificación de SMODELS que puede ser usada para obtener los answer sets preferidos bajo la semántica de Disyunción Ordenada llamado PSMODELS, y el Logics Workbench (LWB)[4] que nos permite trabajar con lógica Intucionista [38].

[1] http://www.dbai.tuwien.ac.at/proj/dlv/
[2] http://www.tcs.hut.fi/Software/smodels/
[3] http://www.dbai.tuwien.ac.at/proj/dlv/K/
[4] http://www.lwb.unibe.ch/about/index.html

1.2.2 Programas-RC y Programas Abductivos

Las reglas para restaurar consistencia (reglas-RC) son las reglas que se agregan a los programas lógicos disyuntivos estándares, pero se aplican solamente cuando las reglas regulares (las reglas estándar) en el programa conducen a inconsistencia. Una regla-RC es escrita como $\alpha \overset{+}{\leftarrow} \beta$, lo cual se lee intuitivamente como: *Si el programa es inconsistente, entonces asumimos* $\alpha \leftarrow \beta$. *En otro caso, ignore la regla.* Recordamos el siguiente ejemplo presentado en [3], puesto que ilustra claramente el uso de las reglas-RC. En este ejemplo r_1 denota el nombre de la regla-RC.

$$a \leftarrow \neg b.$$

$$\neg\, a.$$

$$r_1 : b \overset{+}{\leftarrow} .$$

Las primeras dos reglas son *reglas regulares* y la tercera regla es una regla-RC. Esta clase de programas será llamada CR-programas. Podemos ver que el programa sin la regla-RC es inconsistente. Sin embargo, si la regla-RC se utiliza entonces se restaura la consistencia, así el answer set $\{\neg a, b\}$ es obtenido.

La semántica de los programas-RC se define en términos de los *answer sets mínimos generalizados*. En [3] los autores dan una traducción de programas-RC a programas lógicos abductivos, que llaman *hard reduct*, y la semántica se define como los answer sets mínimos generalizados del hard reduct del programa.

Las definiciones siguientes se encuentran en [3]:

Definition 1.1. [3] Un *programa lógico abductivo* es un par $\langle P, A\rangle$ donde P es un programa arbitrario y A un conjunto de átomos, llamado *abducibles*.

Definition 1.2. [3] Sea $\langle P, A\rangle$ programa lógico abductivo y $\Delta \subseteq A$. Decimos que $M(\Delta)$ es un *answer set generalizado* de P con respecto a A sii $M(\Delta)$ es un answer set de $P \cup \Delta$.

Definition 1.3. [3] Sean $M(\Delta_1)$ y $M(\Delta_2)$ dos answer sets generalizados de P con respecto a A. Definimos $M(\Delta_1) < M(\Delta_2)$ sii $\Delta_1 \subset \Delta_2$ (orden por inclusión de conjuntos).

Definition 1.4. Sea P un programa y sea A un conjunto de átomos. $M(\Delta)$ es un *answer set mínimo generalizado* de P con respecto a A y el orden $<$ sii $M(\Delta)$ es un answer set generalizado de P y es mínimo con respecto al orden por inclusión de conjuntos.

Finalmente, el ejemplo siguiente ilustra cómo la semántica de los programas-RC se define en términos de los *answer sets mínimos generalizados*.

Example 1.1. Sea $P_1 = \{p \leftarrow \neg q.\ \ r \leftarrow \neg s.\ \ q \leftarrow t.\ \ s \leftarrow t.\ \ \leftarrow p,\ r\}$ el programa del Ejemplo 4 introducido en [3]. Podemos ver que el programa P_1 no tiene answer sets. Sin embargo, para restaurar consistencia, podemos definir un Programa-RC que agrega algunas reglas-RC. Por ejemplo, definamos el Programa-RC $P_1' = P1 \cup CR_1$ donde $CR_1 = \{r_1 : q \overset{+}{\leftarrow}, r_2 : s \overset{+}{\leftarrow}, r_3 : t \overset{+}{\leftarrow}\}$. Como mencionamos, la semántica de P_1' está definida en términos de los *answer sets mínimos generalizados*. Entonces, traduciendo P_1' en un *programa lógico abductivo* obtenemos $\langle P_1, A\rangle$ donde el conjunto de *abducibles* es $A = \{q, s, t\}$. Podemos verificar que los answer sets mínimos generalizado de P_1 son $\{q, r\}$, $\{s, p\}$ y $\{t, q, s\}$. □

1.2.3 Programas Lógicos con Disyunción Ordenada

En [7], Brewka introdujo los Programas Lógicos con Disyunción Ordenada, abreviados como LPODs. Para definir esta clase de programas, el conectivo $\times$, llamado *disyunción ordenada*, se agrega al sistema de conectivos para fórmulas proposicionales presentados en la Sección 1.2.1. La intuición detrás de los LPODs es expresar conocimiento por falla

con conocimiento sobre preferencias de una manera simple y elegante [9]. Como hemos mencionado en la Sección 1.2.1, en este trabajo estamos considerando solamente un tipo de negación, negación como falla, pero esto no afecta la validez de los resultados dados en [7]. Formalmente, un programa lógico con disyunción ordenada es un conjunto de reglas de la forma:

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k \tag{1.1}$$

donde C_i, A_j y B_l son todos átomos sin varables. $C_1 \ldots C_n$ son normalmente llamados las opciones de una regla y su lectura intuitiva es la siguiente: Si el cuerpo es verdadero y C_1 es posible, entonces C_1; si C_1 no es posible, entonces C_2; $\ldots$; si ninguno de $C_1, \ldots, C_{n-1}$ es posible entonces C_n. Algunos casos especiales se pueden precisar, $n = 0$ (restricciones), $n = 1$ (programas extendidos), $m = k = 0$ (hechos). La semántica de los programas lógico con disyunción ordenada será presentada en la Sección 1.3.1 donde una extensión de los programas lógicos con disyunción ordenada se presenta. Actualmente, hay una modificación de SMODELS[5] que se puede utilizar para el cálculo del los answer set preferidos de los programas lógicos con disyunción ordenada llamada PSMODELS[6] [9].

1.2.4 Planificación en Answer Sets

En un problema del planificación, estamos interesados en buscar una secuencia de acciones que conduzca de un estado inicial dado a un estado que define la meta. Para especificar totalmente un problema del planificación, necesitamos indicar que acciones se permiten en un plan. La planificación en Answer Sets usa lenguajes de acciones para especificar los efectos de acciones y en particular son usados para modelar problemas

[5] http://www.tcs.hut.fi/Software/smodels/
[6] http://www.tcs.hut.fi/Software/smodels/priority/

de planificación [16]. Algunos de estos lenguajes de acción son $\mathcal{A}, \mathcal{B}$, y $\mathcal{C}$ [16]. Por otra parte, un problema del planificación especificado en estas clases de lenguajes tiene una codificación natural y fácil en Answer Set Programming.

Languaje $\mathcal{A}$. El alfabeto del lenguaje $\mathcal{A}$ consiste de dos conjuntos no vacíos y disjuntos de símbolos **F** llamado el conjunto de fluentes, y **A** llamado el conjunto de acciones. Intuitivamente, un fluente expresa una característica de un objeto en un mundo, y forma parte de la descripción de los estados del mundo. Una *literal fluente* es un fluente o un fluente precedido por $\sim$. Un *estado* σ es un conjunto de fluentes.

Decimos que un fluente f se cumple en un estado σ si $f \in \sigma$. Decimos que un fluente literal $\sim f$ se cumple en σ si $f \notin \sigma$. Cuando las acciones son ejecutadas exitosamente cambian el estado del mundo. Las situaciones son representaciones de la historia de la ejecución de acciones. La situación inicial $[a_n, \ldots, a_1]$ corresponde a la historia donde la acción a_1 se ejecuta en la situación inicial, seguida por a_2, y así hasta a_n. Hay una relación simple entre situaciones y estados. En cada situación s algunos fluentes son verdaderos y algunos otros son falsos.

El lenguaje $\mathcal{A}$ se puede dividir en tres sub-languajes: *lenguaje para descripción del dominio*, *lenguaje de observación* y *lenguaje de consulta*. Dado un problema del planificación, éste puede ser expresado en una lenguaje de acciones por medio de un triple (D, O, G) tal que D es la descripción del dominio, O es el conjunto de observaciones y G es el conjunto de condiciones de la meta. Por otra parte, es posible definir una codificación en Answer Sets de este problema de planificación especificado en un lenguaje de acciones, denotada como $\pi(D, O, G, l)$, donde l es un número entero positivo que corresponde a la longitud del plan, es decir, el número de las acciones que se deben realizar para alcanzar la meta.

Entonces, es posible obtener la solución del problema de planificación (los planes) de los answer sets de $\pi(D, O, G, l)$. La idea básica es que decidimos sobre una longitud

del plan l para enumerar ocurrencias de las acciones hasta los puntos del tiempo que corresponden a esa longitud, y codificar las metas como restricciones acerca del tiempo $l+1$, tales que los answer sets que codifican las ocurrencias de las acciones que no satisfacen la meta en el tiempo $l+1$ son eliminados. Así cada uno de los answer sets que sobreviven a las restricciones corresponde a un plan. Es importante mencionar que en la codificación en Answer Sets de los problemas de planificación expresados en lenguaje $\mathcal{A}$, si f es un literal fluente negativa $\sim g$ entonces f será codificada como $neg(g)$, donde, $neg(g)$ corresponde la negación clásica.

Los detalles de estos sublenguajes y cómo obtener la codificación de un problema del planificación de una especificación en un lenguaje de acciones pueden encontrarse en [16, 4]. Presentamos solamente un ejemplo que muestra cómo especificar el problema simple del planeación del mundo de los bloques en lenguaje $\mathcal{A}$ que proviene de [4].

Example 1.2. El problema de planificación de los cubos puede ser expresado en lenguaje $\mathcal{A}$ como sigue:

1. Clase: $block(X)$. En el resto de este ejemplo, X y Y varían en la clase $block$.

2. Fluentes: $on(X,Y), ontable(X), clear(X), holding(X), handempty$

3. Acciones: $pick\_up(X), put\_down(X), stack(X,Y), unstack(X,Y)$

4. Condiciones de ejecución, denotadas por D:
   executable $pick\_up(X)$ **if** $clear(X), ontable(X), handempty.$
   executable $put\_down(X)$ **if** $holding(X)$.
   executable $stack(X,Y)$ **if** $holding(X), clear(Y)$.
   executable $unstack(X,Y)$ **if** $clear(X), on(X,Y), handempty.$

5. Proposiciones de efecto.
   $pick\_up(X)$ **causes** $\sim ontable(X), \sim clear(X), holding(X), \sim handempty.$

$put\_down(X)$ **causes** $ontable(X), clear(X), \sim holding(X), handempty.$

$stack(X,Y)$ **causes** $\sim holding(X), \sim clear(Y), clear(X), handempty, on(X,Y).$

$unstack(X,Y)$ **causes** $holding(X), clear(Y), \sim clear(X), \sim handempty, \sim on(X,Y).$

6. Considerando la clase *block* como $\{a, b, c, d\}$ las condiciones iniciales, denotadas por I, son descritas como:
   $initially(clear(a)).$ $initially(clear(c)).$ $initially(clear(d)).$ $initially(handempty).$
   $initially(ontable(c)).$ $initially(ontable(b)).$ $initially(on(a,b)).$ $initially(on(d,c)).$

7. Las condiciones de meta, denotadas por G, son el conjunto de literales
   $\{on(a,c), on(c,b), ontable(d)\}$. □

El Lenguaje $\mathcal{B}$ resulta cuando el Lenguaje $\mathcal{A}$ es extendido para permitir *proposiciones de causa estáticas*. Usando proposiciones de causa estáticas podemos expresar conocimiento acerca de los estados en base a los estados que son posibles o validos. Este conocimiento puede ser usado indirectamente para inferir efectos de acciones, ejecutabilidad de las acciones o ambas. Más detalles acerca de proposiciones de causa estáticas y codificación de problemas expresados en Lenguaje $\mathcal{B}$ pueden ser encontrados en [16, 4].

1.2.5 Un lenguaje para preferencias de planes: Lenguaje $\mathcal{PP}$

El lenguaje $\mathcal{PP}$ [43] permite la especificación de preferencias del usuario. Ademas $\mathcal{PP}$ cuenta con una implementación en programación lógica, basada en Answer Set Programming. El lenguaje $\mathcal{PP}$ permite especificar preferencias entre posibles planes y también preferencias temporales sobre los planes. Las preferencias que representan tiempo son expresadas usando los conectivos temporales *next, always, until* y *eventually.* Hay tres clases de preferencias en $\mathcal{PP}$ [43] *deseos básicos*, *preferencias atómicas* y *preferencias generales* . En este trabajo estudiamos las dos primeras clases de preferencias

porque ellas fueron útiles para expresar preferencias en los planes de evacuación.

Un *deseo básico*, denotado como φ, es una fórmula de $\mathcal{PP}$ que expresa una preferencia sobre una trayectoria con respecto a la ejecución de una cierta acción o con respecto a los estados que la trayectoria llega cuando se ejecuta una acción.

Una *preferencia atómica* representa un orden entre fórmulas de *deseo* básico. Indica que las trayectorias que satisfacen φ_1 son preferibles a los que satisfagan φ_2, etc. Claramente, las fórmulas de deseo básico son casos especiales de las preferencias atómicas.

1.3 Contribución

Recordemos que el objetivo de nuestro trabajo es investigar y evaluar las capacidades de Answer Set Programming para representar situaciones del desastre con el fin de apoyar en la definición de planes de la evacuación. Para ello comenzamos a analizar cómo la información geográfica de la zona de desastre a modelar se puede traducir a un formato que Answer Sets es capaz de manejar. Continuamos estudiando y aplicando diversos enfoques en Answer Sets que son útiles para definir los planes de la evacuación, tales como, Programas-RC, Programas Lógicos con Disyunción Ordenada, planificación en Answer Sets, el lenguaje para preferencias de planes y los answer sets mínimos generalizados. Específicamente, definimos la extensión de Programas Lógicos con Disyunción Ordenada y vimos su utilidad. Ademas se presento cómo y porqué los enfoques en Answer Sets mencionados se aplican para definir planes de la evacuación y también se presentaron algunas debilidades de estos enfoques que este trabajo trata.

1.3.1 Programas Lógicos con Disyunción Ordenada Extendidos

En [7] la cabeza de las reglas con disyunción ordenada es definida en términos de literales sin variables. En esta sección, la cabeza y el cuerpo para reglas con disyunción ordenada extendidas son definidos en términos de formulas porposicionales bien formadas. Consideramos que una sintaxis más amplia para estas reglas podría darnos algunas ventajas. Por ejemplo, el uso de expresiones jerarquizadas podría simplificar la tarea de escribir programas lógicos y mejorar su legibilidad puesto que podría permitirnos escribir reglas más cortas y de una manera más natural.

Definition 1.5. Una *regla con disyunción ordenada extendida* es una formula proposicional bien formada como se definió en la Sección 1.2.1, o una formula de la forma:

$$f_1 \times \ldots \times f_n \leftarrow g \tag{1.2}$$

donde $f_1, \ldots, f_n, g$ son formulas proposicionales bien formadas. Un *programa con disyunción ordenada extendido* es un conjunto finito de reglas con disyunción ordenada extendida. □

Las fórmulas $f_1 \ldots f_n$ generalmente son llamadas las opciones de una regla y su lectura intuitiva es como sigue: si el cuerpo es verdadero y f_1 es posible, entonces f_1; si f_1 no es posible, entonces f_2; $\ldots$; si ninguno de $f_1, \ldots, f_{n-1}$ es posible entonces f_n. El caso particular donde toda f_i es una literal y g es una conjunción de literales corresponde a los programas con disyunción ordenada originales según lo presentado por Brewka en [7], y que llamaremos *programas con disyunción ordenada estándar*. Recordamos que los programas con disyunción ordenada estándar de Brewka utilizan el conectivo para la negación fuerte.

Aquí consideraremos solamente un tipo de negación (negación como falla) pero esto no afecta los resultados dados en [7]. Si $n = 0$ la regla es además una restricción, es decir, $\bot \leftarrow g$. Si $n = 1$ es una regla extendida y si $g = \top$ la regla es un hecho y se puede escribir como $f_1 \times \ldots \times f_n$. Ahora, presentamos la semántica de los programas con disyunción ordenada extendidos. La mayoría de las definiciones presentadas aquí se toman de [7, 9].

Definition 1.6. [7]Sea $r := f_1 \times \ldots \times f_n \leftarrow g$ una regla con disyunción ordenada extendida. Para $1 \leq k \leq n$ la *k-ésima opción de r* es definida como sigue: $r^k := (f_k \leftarrow g, \neg f_1, \ldots \neg f_{k-1})$.

Definition 1.7. [7] Sea P un programa con disyunción ordenada extendido. P' es un *programa particionado de* P si es obtenido por reemplazar cada regla $r := (f_1 \times \ldots \times f_n \leftarrow g)$ en P por una de sus opciones $r^1, \ldots, r^k$. M es un answer set de P sii es un answer set[7] de un programa particionado P' de P. □

Definition 1.8. [7]Sea M un answer set de un programa con disyunción ordenada extendido P y sea $r := (f_1 \times \ldots \times f_n \leftarrow g)$ una regla de P. Definimos el *grado de satisfacción de r con respecto a M*, denotado por $deg_M(r)$, como sigue:
—si $M \cup \neg(\mathcal{L}_P \setminus M) \nvdash_{\mathrm{I}} g$, entonces $deg_M(r) = 1$.
— si $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} g$ entonces $deg_M(r) = min_{1 \leq i \leq n} \{i \mid M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} f_i\}$. □

El teorema siguiente indica la relación entre los answer sets de un programa con disyunción ordenada extendido y el grado de satisfacción de cada regla en el programa.

Theorem 1.2. *[7] Sea P un programa con disyunción ordenada extendido. Si M es un answer set of P entonces M satisface todas las reglas en P con algún grado.*

[7] Notemos que debido a que no estamos considerando negación fuerte, no hay posibilidad de tener answer sets inconsistentes.

Definition 1.9. [9] Sea P un programa con disyunción ordenada extendido y M un conjunto de literales. Definimos $S^i_M(P) = \{r \in P \mid deg_M(r) = i\}$.

Con las definiciones anteriores introdicumos dos tipos diferentes de relaciones de preferencia entre los answer sets de un programa con disyunción ordenada extendido: preferencia basada en inclusión de conjuntos y preferencia basada en la cardinalidad de conjuntos.

Definition 1.10. [9] Sea M y N answer sets de un programa con disyunción ordenada extendido P. Entonces decimos que M es *preferido por inclusión* a N, denotado como $M >_i N$, sii hay un i tal que $S^i_N(P) \subset S^i_M(P)$ y para todo $j < i$, $S^j_M(P) = S^j_N(P)$. Decimos que M es *preferido por cardinalidad* a N, denotado como $M >_c N$, sii hay un i tal que $|S^i_M(P)| > |S^i_N(P)|$ y para todo $j < i$, $\left|S^j_M(P)\right| = \left|S^j_N(P)\right|$. □

Definition 1.11. [9] Sea M un answer set de un programa con disyunción ordenada extendido P. M es un *answer set preferido por inclusión* de P si no hay algún answer set M' de P, $M \neq M'$, tal que $M' >_i M$. M es un *answer set preferido por cardinalidad* de P si no hay M' answer set de P, $M \neq M'$, tal que $M' >_c M$. □

1.3.2 Answer Set Mínimos Generalizados y Disyunción Ordenada

Ahora, definimos una caracterización de los answer set mínimos generalizados de un programa en términos de un programa con disyunción ordenada estándar. Este resultado teórico prueba que ambos programas abductivos y los programas con reglas-RC se pueden representar correctamente usando disyunción ordenada.

Definition 1.12. Sea $\langle P, A\rangle$ un programa lógico abductivo. Entonces, definimos una traducción a un programa con disyunción ordenada estándar, denotado por $ord(P, A)$,

como sigue: Primero, para alguna literal $a \in A$ la regla r_a es definida como $a^\bullet \times a$, donde $a^\bullet$ es una literal que no ocurre en el programa original. Definimos $ord(P, A) = P \cup \{r_a \mid a \in A\}$. □

Lemma 1.1. *$M \cap \mathcal{L}_P$ es un answer set generalizado del programa lógico abductivo $\langle P, A\rangle$ sii M es un answer set de $ord(P, A)$.*

El teorema siguiente prueba la validez de nuestra traducción para la semántica de programa lógicos abductivos.

Theorem 1.3. *Sea $\langle P, A\rangle$ un programa lógico abductivo y M un conjunto de átomos. $M \cap \mathcal{L}_P$ es un answer set mínimo generalizado de $\langle P, A\rangle$ sii M es un answer set preferido por inclusión de $ord(P, A)$.*

1.3.3 Preferencias en términos de programas con Disyunción Ordenada Extendidos

Podemos especificar un orden entre los answer sets de un programa con disyunción ordenada extendido con respecto a una lista de átomos.

Definition 1.13. Sea P un programa normal. Sean M y N dos answer sets de P. Sea $C = [c_1, c_2, \ldots, c_n]$ un conjunto ordenado de formulas. El answer set M *es preferido al answer set N con respecto a $C \cup P$* (denotado como $M <_{C \cup P} N$) si

- existe $i = min(1 \leq k \leq n)$ tal que $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} c_i$ y $N \cup \neg(\mathcal{L}_P \setminus N) \nvdash_{\mathrm{I}} c_i$; y
- para todo $j < i$,

  $M \cup \neg(\mathcal{L}_P \setminus M) \vdash_{\mathrm{I}} c_j$ y $N \cup \neg(\mathcal{L}_P \setminus N) \vdash_{\mathrm{I}} c_j$ ó

  $M \cup \neg(\mathcal{L}_P \setminus M) \nvdash_{\mathrm{I}} c_j$ y $N \cup \neg(\mathcal{L}_P \setminus N) \nvdash_{\mathrm{I}} c_j$. □

Proposition 1.1. *Sea $C = [c_1, c_2, \ldots, c_n]$ un conjunto ordenado de formulas; entonces $<_{C \cup P}$ es un orden parcial.* □

Un answer set M de un programa normal P *es el más preferido* si no hay otro answer set N de P que sea preferido a M.

Example 1.3. Sea $C = [b, c]$ un conjunto ordenado de átomos. Sea P el siguiente programa normal: $\{a \leftarrow .\quad c \leftarrow \neg b.\quad b \leftarrow \neg c.\}$. Podemos verificar que P tiene dos answer sets, $\{a, b\}$ y $\{a, c\}$. También que el answer set $\{a, b\}$ *es preferido al answer set* $\{a, c\}$ *con respecto a* C, i.e., $\{a, b\} <_{C \cup P} \{a, c\}$ y que $\{a, b\}$ es también *el más preferido answer set.* □

Definition 1.14. Sea P un programa normal y $C = [c_1, c_2, \ldots, c_n]$ un conjunto ordenado de átomos tal que $C \subseteq \mathcal{L}_P$. Definimos una *regla con disyunción ordenada extendida definida de* C, denotada como r_C, como sigue: $r_C := (\neg\neg c_1 \times \neg\neg c_2 \times \ldots \times \neg\neg c_n \times all\_pref)$ tal que $all\_pref \notin \mathcal{L}_P$. □

Lemma 1.2. *Sea P un programa normal y $C = [c_1, c_2, \ldots, c_n]$ un conjunto ordenado de átomos tal que $C \subseteq \mathcal{L}_P$. Sea r_C la regla con disyunción ordenada extendida definida de C. Entonces M es un answer set preferido por inclusión de $P \cup r_C$ sii $(M \cap \mathcal{L}_P)$ es el mas preferido answer set con respecto a $C \cup P$.* □

Example 1.4. Consideremos otra vez el programa P del Ejemplo 1.3 y el conjunto ordenado de átomos $C = [b, c]$. Entonces, $P \cup r_C$ es el siguiente programa $\{a \leftarrow .\quad c \leftarrow \neg b.\quad b \leftarrow \neg c.\quad \neg\neg b \times \neg\neg c \times all\_pref.\}$. Podemos verificar que $\{a, b\}$ es el mas preferido answer set con respecto a $C \cup P$ y que es también un answer set preferido por inclusión de $P \cup r_C$. □

Vale la pena mencionar que ni ejecutando *PSMODELS*[8] [9] o siguiendo la definición

[8] http://www.tcs.hut.fi/Software/smodels/priority/

dada por Brewka [7] para disyunción ordenada podemos obtener los answer sets preferidos por inclusión de conjuntos para programas con disyunción ordenada extendidos. La razón es que la definición dada por Brewka para disyunción ordenada tiene restricciones sintácticas.

Sin embargo, en particular cuando este programa tiene reglas de disyunción ordenada extendidas usando literales negativas negadas podemos fácilmente traducir este programa a un programa con disyunción ordenada estándar y entonces usar *PSMODELS* para obtener los answer sets preferidos [46]. En la siguiente definición y lema los átomos $a^\bullet$, a°, no ocurren en el programa original P.

Definition 1.15. Sea $\neg\neg a$ una literal con doble negación como falla. Definimos el conjunto de reglas con disyunción ordenada asociadas a $\neg\neg a$ como sigue:
$R(\neg\neg a) := \{ \leftarrow \neg a, a^\bullet. \quad a^\bullet \leftarrow \neg a^\circ. \quad a^\circ \leftarrow \neg a. \quad \leftarrow a, a^\circ. \ \}.$ □

Lemma 1.3. *Sea P programa normal y sea $C = [c_1, c_2, \ldots, c_n]$ un conjunto ordenado de átomos tal que $C \subseteq \mathcal{L}_P$. Sea $C^\bullet = [c_1^\bullet, c_2^\bullet, \ldots, c_n^\bullet]$ un conjunto ordenado de átomos tales que $\{c_i^\bullet \in C \mid 1 \leq i \leq n\} \cap \mathcal{L}_P = \emptyset$. Sea r_C la regla con disyunción ordenada extendida definida de C. Sea $r_C^\bullet$ la regla con disyunción ordenada extendida definida de $C^\bullet$. Sea $A = \bigcup_{c_i \in C \text{ and } 1 \leq i \leq n} R(\neg\neg c_i)$. Entonces M es un answer set preferido por inclusión de $P \cup r_C^\bullet \cup A$ sii $M \cap \mathcal{L}_P$ es un answer set preferido por inclusión de $P \cup r_C$.* □

1.3.4 Información geográfica

La mayoría de la información necesaria para modelar el problema del plan de la evacuación se debe obtener de la información geográfica. Definimos un procedimiento para construir el conocimiento acerca de la zona del peligro a partir de la información geográfica. Este procedimiento es resultado de nuestra propia experiencia al haber trabajado con esta clase de datos. Vale la pena mencionar que en este trabajo probamos

todos nuestros resultados con una pequeña parte de los datos geográficos verdaderos sobre zona del peligro de Popocatépetl del volcán, puesto que los datos verdaderos de toda esta zona casi fueron obtenidos al terminar este trabajo. El procedimiento puede ser descrito en terminos de los siguientes pasos: Extracción de la información descriptiva [1] que forma parte de la información geográfica con ayuda de una herramienta que maneja este tipo de información, reparación de inconsistencias, y finalmente añadir información descriptiva adicional.

1.3.5 Problema de plan de evacuación

Para obtener los planes de la evacuación, comenzamos a modelar el problema de los planes de evacuación usando Answer Set Planning.

Normalmente, en una zona del riesgo se definen rutas evacuación por las autoridades en la zona del riesgo. Cada ruta de evacuación comienza en un conjunto de lugares en riesgo, atraviesa otros lugares de menor riesgo y llega a lugares fuera de riesgo. Algunas veces, el lugar fuera de riesgo corresponde a un refugio. Cada refugio tiene bastantes provisiones para la gente. Sin embargo, algunos peligros que pueden acompañar un desastre pueden resultar en el bloqueo de parte de las rutas de evacuación predefinidas. Entonces es necesario definir planes alternativos de evacuación. El *problema de definir planes de evacuación alternativos* (AEP-problema) puede ser enunciado como sigue:

> *Hay un conjunto de rutas de evacuación predefinidas para la gente que vive en la zona del peligro. Cada ruta de evacuación predefinida puede tener varios puntos iniciales, pero un solo punto final. En caso de que una parte de una ruta de evacuación predefinida sea inaccesible, entonces los evacuados deben buscar una trayectoria alternativa. Esta trayectoria alternativa puede pertenecer o no a otra ruta de la evacuación. Si no pertenece a una ruta de la evacuación entonces se prefiere, en orden que descendente, llegar a un cierto punto que pertenece a una ruta de la evacuación o a un refugio o a un lugar fuera de riesgo.*

Presentamos dos soluciones al problema de definir planes de evacuación alternativos, ambos usan Answer Set Planning y algunos enfoques en Answer Sets para preferencias. Aplicar preferencias resulta ser una manera muy natural y eficaz de obtener una solución apropiada para algunos problemas. Por ejemplo, en caso de tener un conjunto de deseos acerca de un plan de evacuación que quisiéramos satisfacer y al mismo tiempo no todos ellos pueden ser satisfechos simultáneamente. Otro ejemplo es cuando dado un problema del planificación, se podría obtener un alto número de soluciones. Por tanto, es necesario expresar preferencias para elegir alguno de los planes posibles de evacuación. Las soluciones presentadas en este trabajo se dirigen a la especificación de tales preferencias entre planes factibles. Hemos explorado dos enfoques en answer sets: *reglas-RC* y el Lenguaje $\mathcal{PP}$.

Usando reglas-RC

Ampliamos un problema de planificación modelado usando Answer Set Planning agregandole reglas-RC [47, 50]. La idea es obtener los planes de evacuación alternativos aprovechando la definición de una regla-RC y utilizar estas reglas sólo si no hay manera de obtener un plan de evacuación cuando parte de la ruta predefinida se bloquea. Por lo tanto, las reglas-RC son definidas sobre las acciones usadas para modelar el problema. En caso de que una regla-RC se aplique se restaura la consistencia permitiendo obtener los planes de evacuación alternativos. En particular, la regla-RC que agregamos es la siguiente:

$$r_2 : action(travel(B, P, Q, R')) \stackrel{+}{\leftarrow} bus(B), road(P, Q, R').$$

La intuición de la regla-RC r_2 es que es posible viajar de P a Q si hay un segmento de camino de P a Q. En contraste a la acción regular *travel* definida en $\pi(D, O, G, l)$, esta regla-RC no checa si el camino de P a Q pertenece a una ruta de evacuación,

es decir, no es importante si R' es igual a cero o no. De esta forma, r_2 nos permite encontrar los planes de evacuación alternativos. Véanse los ejemplos A.1, A.2, y A.3 en el Apéndice A que muestra como podemos usar las reglas-RC para obtener los planes de evacuación alternativos. Hacemos hincapié en que dado un programa con reglas-RC que obtiene los planes de evacuación alternativos, podemos traducirlo de acuerdo a la Definición 1.12 para obtener un programa con disyunción ordenada estándar y entonces usar PSMODELS[9] para obtener los answer sets preferidos que corresponden a los planes de evacuación alternativos.

Usando el Lenguaje $\mathcal{PP}$

Como hemos visto, usar reglas-RC permite obtener los planes de evacuación alternativos sin embargo, estos planes alternativos no consideran ninguna otra característica de la trayectoria que ellos siguen. Por lo tanto, consideramos el lenguaje $\mathcal{PP}$ para expresar preferencias sobre los planes alternativos a diversos niveles. Decidimos utilizar $\mathcal{PP}$ porque nos permite expresar preferencias en términos del tiempo. Por ejemplo, *siempre* se prefiere evacuar a la gente en riesgo siguiendo las rutas de evacuación predefinidas. Sin embargo, si *eventualmente* parte de las rutas de evacuación se bloquea entonces los evacuados viajarán fuera de la ruta de evacuación *hasta* que ellos lleguen a un refugio. También demostramos que las reglas con disyunción ordenada y con literales negativas negadas podrían ser útiles para permitir una codificación más simple y más fácil de obtener los planes preferidos con respecto a una preferencia expresada en la lenguaje $\mathcal{PP}$. El ejemplo A.4 en el apéndice A muestra el uso de la lenguaje $\mathcal{PP}$ para expresar las preferencias entre los planes de la evacuación.

[9] http://www.tcs.hut.fi/Software/smodels/priority/

1.3.6 Extendiendo el Lenguaje $\mathcal{PP}$

Mientras que utilizamos $\mathcal{PP}$ para expresar preferencias nos dimos cuenta que hay algunas preferencias que no se pueden expresar en una manera simple y natural puesto que estas preferencias resultan ser muy grandes. Por lo tanto, para tener una representación más natural de esta clase de preferencias definimos el lenguaje $\mathcal{PP}^{par}$. $\mathcal{PP}^{par}$ es una extensión del lenguaje $\mathcal{PP}$ donde los conectivos proposicionales y temporales permiten representar de manera compacta las preferencias que tienen una característica común en particular. Véase el ejemplo A.5 del Apéndice A.

Además hemos considerado el hecho de que el lenguaje $\mathcal{PP}$ podría tomar ventaja de todo el desarrollo teórico de la Lógica Proposicional Lineal Temporal (LTL) para expresar preferencias. Por lo tanto hemos analizado como podríamos establecer la relación entre el lenguaje $\mathcal{PP}$ y LTL.

1.3.7 Contenido Semántico

Hemos presentado la noción de *Contenido Semántico* de un programa como un punto de vista alternativo para obtener diferentes semánticas en answer sets de un programa. En particular, mostramos como obtener los *answer sets estándar*, los *answer sets generalizados*, los *answer sets mínimos generalizados* y una nueva semántica que llamamos *answer sets parciales*. Finalmente, proponemos hacer uso de los answer sets parciales para obtener planes de evacuación parciales.

1.4 Conclusiones

En esta tesis hemos propuesto investigar y evaluar las capacidades deAnswer Set Programming para representar situaciones del desastre con el objeto de dar la ayuda en

definir planes de la evacuación. Llegamos a la conclusión que usando Answer Set Programming es posible representar situaciones del desastre y sobre todo aprovechar sus capacidades y los diversos enfoques en Answer Sets que existen actualmente para obtener los planes de evacuación. Además, mostramos cómo y porqué los enfoques en Answer Sets estudiados son útiles para definir los planes de evacuación. Finalmente, presentamos algunas debilidades de estos enfoques que tratamos en este trabajo.

Un asunto interesante para la investigación a futuro es el uso de algunos enfoques Answer Sets dirigidos a hacer la generación de los answer sets más eficaz en la generación de los planes de evacuación. Por ejemplo con respecto a sus requisitos de la memoria podrían usarse mecanismos mas eficaces como el que se propone en [5]. Otra dirección que podría investigarse es modelar dominios dinámicos usando Answer Sets para generar los planes de evacuación donde las características relevantes de los eventos exógenos (por ejemplo una explosión o los flujos de la lava) que ocurren en una zona del peligro y la información de dicha zona sean consideradas. Los enfoques en Answer Sets usados en [3] para modelar sistemas dinámicos y en [10] para modelar flujos de lava podían ayudarnos en esta dirección. Finalmente, sería interesante ampliar nuestro uso del Contenido Semántico para encontrar algunas otras variantes de la semántica de answer set, tales como los programas con disyunción ordenada y entonces, analizar su uso para obtener los planes de la evacuación. Por ejemplo en [48] se ilustran algunos ejemplos pequeños usando los answer set parciales definidos en términos de su contenido semántico.