CAPITULO I

Preliminares

La planeación como un problema en lógica data desde McCarthy en los años cincuenta, el descubrimiento del método de resolución de Robinson puso las bases para la planeación deductiva en [Green 1969] y el cálculo de situaciones mencionado en [McCarthy y Hayes 1969]. Sin embargo, debido a los defectos como el problema del frame, la planeación deductiva perdió atención, mientras los operadores STRIPS definidos en [Fikes y Nilsson 1971], un híbrido entre lógica y computación procedural, y sus variantes estaban ganando importancia. Entonces, por un largo periodo ningún otro sistema de planeación lógica relacional se exploró [Eiter et. al. 2002].

Sin embargo, en los últimos 12 años, la planeación basada en lógica renace y surge en las siguientes líneas de trabajo:

- Las soluciones al problema del frame se han trabajado más a fondo, y la planeación deductiva basada en el cálculo de la situación circunstancial ha sido considerada extensivamente, en particular por el grupo de Toronto, llevando al lenguaje de planeación GOLOG [Levesque et al. 1997]. En paralelo, planeando en el cálculo de evento [Kowalski y Sergot 1986] se ha seguido y se ha empezado de [Eshghi 1988; Shanahan 1989].
- Formulando problemas de planeación como problemas lógicos de satisfactibilidad, propuesto por Kautz y Selman [Kautz y Selman 1992],

permitió resolver grandes problemas de planeación que no podrían ser resueltos por sistemas especializados de planeación, y condujo al eficaz sistema de planeación Caja Negra (BlackBox) [Kautz y Selman 1999]. En el mismo sentido, otros acercamientos redujeron problemas de planeación a tareas computacionales en formalismos lógicos, incluyendo programación lógica [Dimopoulos et al. 1997; Subrahmanian y Zaniolo 1995], chequeo de modelos [Cimatti y Roveri 1999; 2000], y Cuantificación boleana de formulas [Rintanen 1999a].

- Planeación como una tarea en lenguajes basados en lógica para razonamiento acerca de las acciones, que se desarrollaron en el contexto de lógica para la representación de conocimiento y programación lógica. Usando la implementación de esos lenguajes en el sentido de Kautz y Selman, "soluciones satisfactibles" conducen a Causal Calculator (CCALC) [McCain y Tornero 1998; McCain 1999] y el sistema del C-plan [Giunchiglia 2000] que esta basado en el lenguaje de acción C [Giunchiglia y Lifschitz 1998].

En artículos muy influyentes, Lifschitz propone **Answer Set Programming** como una herramienta para resolver el problema, y en particular para planeación [Lifschitz 1999a; 1999b] en este contexto, los problemas de planeación, son formulados en un lenguaje de planeación en un dominio independiente, son mapeados en un programa lógico extendido tal que los answer sets de este programa obtengan las soluciones del problema de planeación. De esta manera, planearon crear lenguajes de descripción de

acciones que soporten expresiones, y el uso de answer sets tal como Smodels o DLV, que permitan resolver problemas de una manera eficaz.

Answer set programming es un paradigma de programación completamente diferente a la programación procedural, en lugar de escribir algoritmos para resolver el problema, el programador describe el problema usando un lenguaje formal y el compilador encuentra una solución al problema.

1.1 SMODELS *[Syrjänen 2000].*

Smodels es un sistema para Answer sets programming. Este consiste de smodels, una eficiente implementación de la semántica de modelos estables para programas lógicos normal, y lparse, un front – end que transforma programas del usuario de tal manera que smodels los entienda.

Un programa en smodels se escribe usando notación estándar de programación lógica. Esto es, los programas están compuestos de átomos y reglas de inferencia. Un átomo representa una demanda acerca del universo del problema y puede ser verdadera o falsa. Las reglas de inferencia son usadas para codificar relaciones entre átomos. Una respuesta al programa es un conjunto de átomos, llamado un modelo estable, esto dice que átomos son verdaderos.

Un programa en Smodels puede tener uno, ninguno o varios modelos estables. Los modelos estables de un programa pueden ser vistos como un conjunto de creencias racionales acerca del programa. Esto es si nosotros pensamos que un programa esta codificado basado en el conocimiento, la

relación entre objetos y un modelos estable es un conjunto de esas cosas en nuestro universo que creemos son verdaderas, entonces nuestras creencias son consistentes y bien fundamentadas. Consistencia significa que no creemos en dos cosas contradictorias y bien fundamentadas significa que tenemos alguna razón para creer. Es decir nosotros no queremos creer que la luna esta hecha de queso a menos que alguien de una teoría coherente del por que la luna es actualmente un enorme producto lácteo.

Smodels consta de dos partes, Smodels y lparse. La primera parte Smodels, es la programación lógica actual que hace el trabajo duro y lparse solo agrega una capa de sintáctica en el tope de este. Smodels es desarrollado en el Laboratorio de Ciencias computacionales teóricas de la Universidad Tecnológica de Helsinki por Patrik Simons, y el lparse fue desarrollado por Tommi Syrjänen.

1.2 DLV *[Faber 2003].*

Datalog es un lenguaje declarativo. Esto significa que el programador no escribe un programa que resuelve algún problema pero en cambio especifica lo que puede parecerse a la solución, y un traductor de Datalog (o Sistema del base de datos deductivo) intenta encontrar la manera de resolver el problema y la propia solución. Esto se hace con reglas y hechos. Los hechos son los datos de la entrada, y pueden usarse reglas para derivar más hechos, y esperanzadamente, la solución del problema dado.

El datalog disyuntivo es una extensión de datalog en que la expresión lógica OR (la disyunción) se permite aparecer en las reglas, esto no se permite en datalog básico.

DLV (datalog con disyunción) es un poderoso sistema de sistema de base de datos deductivo disponible gratuitamente. Es basado en el lenguaje de programación declarativo datalog que es conocido por ser una herramienta conveniente para la representación de conocimiento. Con su extensión disyuntiva, está bien preparado para todos los tipos de razonamiento no monótono, incluyendo diagnóstico y planeación.

DLV es pertinente a dos comunidades. Primero, como se mencionó, es un traductor de bases de datos deductivas y puede verse por consiguiente como una manera de realizar consultas en una base de datos (todo lo que puede hacerse con el lenguaje SQL también puede hacerse con DLV, y más), pero también se describe como un sistema para Answer Set Programming (ASP). Éste es un nuevo paradigma del área de "Razonamiento no monótono" qué permite formular problemas muy complicados de una manera directa y declaratoria. Uno puede llamar a este paradigma aun más declaratorio que la lógica clásica.

DLV cuenta con varios tipos de front- end para abductivos y consistencia basados en el razonamiento de diagnostico (Reiter). Una restricción general

para el razonamiento de diagnostico Reiter's es que las hipótesis pueden ser formuladas únicamente de el predicado "ab". Mas aún varias restricciones son impuestas sobre la teoría de razonamiento de diagnostico.

El resultado de un cálculo de Datalog se llama un modelo. El significado de esto está claro: es una explicación consistente (modelo) del mundo, hasta donde el sistema de Datalog puede derivarlo. Si un programa del datalog es incoherente, es decir, es contradictorio, no hay ningún modelo simplemente.

DLV intenta encontrar todos los modelos del mundo que explican correcta y consistentemente las observaciones hechas. Un modelo asigna un valor de verdad (verdadero o falso) a cada átomo que aparece en el programa, y se escribe como el conjunto de átomos que son verdaderos en un cierto modelo. Cuando todos los átomos son falsos en un modelo, nosotros hablamos sobre un modelo vacío (escrito como {}). Un modelo vacío es muy diferente a encontrar ningún modelo.

Los programas datalog simples tiene un modelo exactamente. En general, sin embargo, los programas de DLV pueden tener cero (como se mencionó) o incluso muchos modelos.

1.3 DLV^k

El sistema **DLV^{k}** fue implementado sobre la lógica de la programación en el sistema DLV, para planeación bajo conocimiento incompleto. nombrado *K* para dar énfasis a que describe transiciones entre los estados de conocimiento en lugar de entre los estados del mundo. Es decir, el lenguaje *C* y muchos otros están basados sobre extensiones de lógica clásica y describen transiciones entre los posibles estados del mundo. Aquí, un estado del mundo es caracterizado por los valores de verdad del numero de fluentes, es decir predicados que describen propiedades relevantes del dominio de discurso, donde cada fluente necesariamente es verdadero o falso. Una acción sólo es aplicable si alguna condición previa (fórmula sobre los fluentes) es verdadera en el estado actual, y ejecutando esta acción cambia el estado actual modificando los valores de verdad de algunos fluentes.

Sin embargo, usualmente los agentes de planeación no tienen una vista completa del mundo. Aun cuando su conocimiento está incompleto, esto es, varios fluentes son desconocidos, ellos deben tomar decisiones, deben ejecutar acciones, y deben razonar en base a su (incompleta) información a la mano.

Por ejemplo, imagine un robot delante de una puerta. Si desconoce que la puerta está abierta, el robot puede decidir retroceder. Alternativamente, podría decidir darse cuenta del estado de la puerta para obtener información completa. Sin embargo, esto requiere que una acción de percepción conveniente este disponible y, pretenciosamente, realmente ejecutable (es decir, el sensor no esta descompuesto). Así, incluso en la presencia de

sensibilidad, algunos fluentes pueden permanecer desconocidos y pueden dejar a un agente en un estado de información incompleta, el lenguaje *K* adopta una vista tri-valuada de fluentes en el que sus valores podrían ser verdad, falso, o desconocido. El lenguaje es muy flexible, y es capaz de modelar transiciones entre los estados del mundo (es decir los estados de conocimiento completo) y de razonamiento sobre ellos como un solo caso.

Comparado con otros lenguajes de planeación similares, *K* es más cercano al estilo de la semántica Answer set [Gelfond y Lifschitz 1991] que a las lógicas clásicas. Permite el uso explícito de negación por falla (default), explota el poder de Answer set para tratar con conocimiento incompleto. A diferencia, de otros lenguajes de acción que permiten estados incompletos [Son y Baral 2001], *K* no adopta posibles estados del conocimiento del mundo y razona sobre los posibles casos para determinar transiciones entre los estados del conocimiento.

1.4 Lenguajes de Acción

Lenguajes de acción son modelos formales de partes del lenguaje natural que son usados para hablar acerca de los efectos de las acciones.

Lenguaje de Acción Descripción A .

Considere un alfabeto de acción proposicional fijo $<\{f,t\},F,A>$.

Definición 1.1 [Gelfond y Lifschitz 1998]**.** En el lenguaje ***A***

(i) una *proposición* es una expresión de la forma

$$A \textbf{ cause } L \textbf{ if } F \qquad (1)$$

donde A es un nombre de acción, L una literal llamada cabeza (head) y F una conjunción de literales (posiblemente vacía).

(ii) Una *descripción de acción* (action description) es un conjunto de proposiciones.

Se denota la conjunción vacía como verdadero (true). En (1), la parte ***if*** *F* puede ser quitada si *F* es true.

En la siguiente definición identificamos una interpretación proposicional con el conjunto de literales que satisface esta interpretación.

Definición 1.2 [Gelfond y Lifschitz 1998]**.** Sea *D* una descripción de acciones en ***A***. El sistema de transición $<S,V,R>$ descrito por *D* esta definido como sigue:

(i) *S* es el conjunto de todas las interpretaciones de *F*

(ii) $V(P,s) = s(P)$

(iii) *R* es el conjunto de todas las triadas $<s,A,s'>$ tal que

$$E(A,s) \text{ Í } s' \text{ Í } E(A,s) \text{ È } s \qquad (2)$$

Donde $E(A,s)$ se posiciona para el conjunto de las cabeceras *L* de todas las proposiciones *A* ***causes*** *L* ***if*** *F* en *D* tal que *s* satisface *F*.

Intuitivamente, *E(A,s)* lista los efectos de la acción *A* ejecutada en el estado *s*. La primera inclusión en (2) requiere que todos esos efectos se cumplan en el estado resultante *s'*. La segunda inclusión dice que en otro caso los valores de las fluentes en *s'* son los mismos que en *s*, así es como la semántica de ***A*** resuelve el problema del frame.

Para cualquier *A* en *s*, esta puede tener a lo mas un estado *s'* que satisface (2), así que cualquier acción descrita en *A* es determinista en cualquier estado.

Lenguaje de Acción Descripción B

El lenguaje ***B*** es una extensión del lenguaje *A* en el que uno puede describir acciones con efectos indirectos. Esto se lleva a cabo introduciendo proposiciones de un nuevo tipo, "leyes estáticas (static laws)". Se entiende por ley estática en *B* como una regla para inferir literales.

Lenguaje de Acción Descripción C

El lenguaje de acción ***C*** es similar a *B* en que sus proposiciones son clasificadas en leyes estáticas y dinámicas. De alguna manera este es más expresivo (aunque estrictamente hablando, este no es un súper conjunto de *B*). Primero, acciones no deterministas y la ejecución de concurrencia pueden ser convenientemente descritas en *C*. Segundo, la inercia no se construye por falla en *C*; decidimos libremente para cada fluente si se postula o no inercia para este.

La vista de causalidad adoptada en *C* distingue entre afirmar que un cierto hecho “se cumple” (holds) y hacer la afirmación fuerte de que esto “es causado” (is caused, o tiene una explicación causal).

En *C* usamos ***default*** para decir que una cierta condición se cumple siempre que esta no haya sido prevenida por ejecutar una acción. La abreviación ***may caused*** ayuda a especificar los efectos de alguna acción no determinista. La abreviación ***always*** nos permite codificar “qualification constraints” en el sentido de [Lin and Reiter, 1994].

Axiomas, Consultas e Historias.

La sintaxis de un lenguaje de acción de consulta esta definido por dos clases de expresiones sintácticas: “axiomas” y “consultas” (queries). La semántica de el lenguaje esta definida por la especificación, para cada sistema de transición *T*, todo conjunto de axiomas G, y toda consulta *Q*, siempre que *Q* sea una consecuencia de G en *T*; simbólicamente, siempre que $\Gamma \models_T Q$.

La semántica de un lenguaje de acción de consulta puede ser previamente definida en términos de “*historias*” que son rutas en la dirección representada por el grafo *T*.

Definición 1.3 [Gelfond y Lifschitz 1998]***:*** Una historia de un sistema de transición <*S,V,R*> de longitud *n* es una secuencia

$s_0, A_1, s_1, \ldots, A_n, s_n$

$(s_0,\ldots,s_n \hat{I} S,\ A_1,\ldots,A_n\ \hat{I} A,\ n\ ^3 0)$ *tal que*

<si-1, Ai, si> ÎR (1 £ i £ n)

Lenguaje de Acción de Consultas P

El lenguaje ***P*** esta diseñado para describir proyección temporal: sus axiomas son observaciones hechas en el estado actual del mundo, y sus consultas (queries) son afirmaciones hipotéticas acerca de los efectos de la ejecución secuencial de acciones en este estado.

Lenguaje de Acción de Consultas Q

En el lenguaje ***Q***, axiomas y consultas son acerca de una secuencia de acciones que tienen que ser ejecutadas.

1.5 Algunas Aplicaciones

En trabajos mas recientes se han desarrollado otros lenguajes como el Lenguaje ***H*** [Watson y Chintabathina, 2003] el cual puede ser visto como una extensión del lenguaje *B*, sin embargo en el pasado por ejemplo un fluent fue normalmente definido como una "propiedad cuyo valor puede cambiar con el tiempo". La definición tradicional no aplica a propiedades cuyo valor cambia continuamente con el tiempo.

Un servicio de bioinformática es un servicio web, i.e, un servicio computacional accesible vía web. Adoptando la vista de considerar servicios web como acciones para aplicación de composición de servicios web, nosotros vemos cada servicio de bioinformática como una acción en calculo de situaciones. Sea *D* el conjunto de acciones representando los servicios

bioinformáticos. Bajo este panorama, el problema de combinar servicios bioinformáticos al desarrollar una aplicación deductiva filogenética es un problema de planeación en *D*. *D* contiene todos las operaciones externas que un usuario de ΦLOG puede usar, cada programa *FLOG P* puede ser compilado en un programa GOLOG en el lenguaje *D* [T.C. Son et.al, 2002].

En el área de ambientes CSCL (Computer Supported Collaborative Learning). Una representación lógica de el modelo aprendiz es propuesto como un conjunto de creencias que un agente lleva a cabo acerca de los intereses y capacidades de su usuario. Basados en esas creencias los agentes aseguran conocimiento y asiste la colaboración entre los aprendices en la comunidad [Ortiz 2003].

Así como estas existen muchas otras aplicaciones como en el área de Cognitive Robotics Logic [Sandewall, 1998], en problemas GIS (Geographic Information System) [Zepeda 2003] pueden hacerse con los fundamentos de Answer Set Progamming. Una aplicación mas es para el modelado de trafico [Turner, et. Al, 2003] en el cual se planten dos problemas de trafico Zoo World y Traffic World los cuales se modelan con Causal Calculator (Ccalc) para probar las posibilidades expresivas del lenguaje *C+*, su principal interés son las expresiones del fragmento “definido” de *C+* que esta implementado en el Causal Calculator (CCalc).