CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
Departamento de Ingeniería Eléctrica
Sección de Computación

Un enfoque de solubilidad en los sistemas de planeación

TESIS que presenta
HÉCTOR SALDAÑA ALDANA
para obtener el grado de
DOCTOR EN CIENCIAS
en la especialidad de **Ingeniería Eléctrica**

Trabajo dirigido por los doctores:
Zděnek Zdráhal y Guillermo Morales L.

México, D.F., septiembre de 1989.

AGRADECIMIENTOS

Primeramente quiero manifestar mi reconocimiento al Dr. Adolfo Guzmán Arenas por su dirección al inicio de mis actividades en el programa de doctorado y por su apoyo total durante su administración como jefe de la Sección de Computación; así como por la revisión exhaustiva y concienzuda que hizo de todos los capítulos del manuscrito original. Por sus sugerencias y comentarios al mismo, los que me permitieron elaborar la versión final, de la cual yo asumo toda la responsabilidad en las interpretaciones y desarrollos presentados.

Al Dr. Zdeněk Zdráhal por su orientación y guía en el desarrollo inicial e intermedio de este trabajo y por la revisión del manuscrito y sus comentarios. Y al Dr. Guillermo Morales Luna por su valiosa dirección en la etapa final del desarrollo de este trabajo y en particular por todas sus sugerencias y guía en la parte teórica.

También agradezco al Dr. Iván Bratko sus comentarios y sugerencias que hizo sobre mi trabajo durante su visita a México en noviembre de 1987. A Peter Clark del Instituto Turing de Edinburgo por su ayuda en la recopilación de citas bibliográficas sobre el tema de los sistemas de planeación y sus indicaciones. Igualmente mi agradecimiento al Dr. Josef Kŏlar por las revisiones e invaluables comentarios que hizo sobre la parte teórica de la tesis.

Agradezco al Centro de Investigación y Estudios Avanzados del IPN el apoyo en bibliotecas, servicios, computadoras, equipos periféricos, etc., tanto del propio Departamento de Ingeniería Eléctrica como del Departamento de Matemáticas. A sus diferentes autoridades, a sus investigadores, técnicos, profesores, secretarias, alumnos y ex-alumnos, que de una u otra forma me ayudaron a lograr este trabajo.

De la misma manera, al Instituto Nacional de Investigaciones Nucleares por el apoyo que me brindo. Y al CONACyT por la beca académica otorgada durante una parte de mi estancia en el CINVESTAV.

Contenido

Lista de Figuras

Pues el Señor es quien da la sabiduría;
la ciencia y el conocimiento brotan de
sus labios.

Proverbios 2:6

Capítulo 1

SISTEMAS DE PLANEACIÓN

1.1 Introducción

El bienestar de la sociedad actual depende muy fuertemente de su desarrollo industrial. Una industria saludable y productiva, está en función de la calidad y el volumen de la producción que tenga, y esto a su vez depende de los métodos de manufactura e ingeniería industrial que se utilicen.

El desarrollo tecnológico actual hace necesario que cada vez más se usen nuevos métodos y técnicas para lograr mejores resultados en la industria. Para ello se ha considerado que las técnicas desarrolladas en el área de la Inteligencia Artificial (IA) pueden ayudar satisfactoriamente en los procesos de planeación y diseño automáticos, en el diagnóstico oportuno, en el análisis de riesgos y en la toma de decisiones.

Las técnicas de la IA se han venido desarrollando desde los años 60's a la fecha y ahora es un reto el utilizarlas para fines prácticos de manufactura e ingeniería industrial así como de otras áreas del conocimiento humano, en donde antes no era posible utilizarlas, por razones tecnológicas.

Algunos enfoques de la IA para resolver problemas concretos han resultado en desarrollos útiles e interesantes en las áreas de visión, robótica y sistemas basados en conocimiento (sistemas expertos y sistemas de planeación).

La planeación automática originalmente se enfocó a desarrollar métodos para la solución de problemas de robots, mediante planes sencillos, esto es, secuencias lineales de acciones para alcanzar metas sencillas en ambientes estáticos.

Los enfoques actuales plantean trabajos de investigación para ampliar aquellos que han sido tradicionales.

Al estar haciendo una revisión de las técnicas y métodos utilizados por los sistemas de planeación, detecté algunos problemas en su funcionamiento, lo cual impedía su aplicación a situaciones del mundo real. Por lo que inicié una investigación, que me condujera a la solución de estos problemas.

Implementé en una microcomputadora un sistema de planeación típico y trabajé con

él algunos ejemplos de planeación. Observé que algunos de los problemas que le planteaba no los podía resolver, a pesar de que sí tenían solución. Encontré que haciendo un replanteamiento del problema era posible resolver muchos de los problemas para los cuales no se pudo encontrar anteriormente una solución y que eran solubles.

Este replanteamiento consiste en hacer una reordenación jerárquica de las partes que describen al problema, de acuerdo a criterios heurísticos.

Tal reordenación se lleva a cabo mediante un algoritmo de jerarquización que está sustentado por toda una formalización matemática desarrollada en este mismo trabajo, útil para describir las funciones y propiedades de los sistemas de planeación. La definición y discusión formal del problema investigado así como la solución que propongo se encuentran en el capítulo 2.

Los programas de cómputo utilizados y las corridas de los experimentos realizados están descritos en el capítulo 3.

En el resto de este capítulo haré un bosquejo del tipo de problemas que me propongo resolver, una presentación de los sistemas de planeación y una reseña histórica de ellos.

¿Cuáles son los problemas ?

Al estar trabajando en la implementación de WARPLAN, que es un planeador no-lineal, conjuntivo e independiente del dominio [Warren74], detecté algunos problemas. En particular, encontré que cuando se especificaba un problema sencillo, el sistema no proporcionaba ninguna solución, a pesar de que el problema dado estaba bien planteado y que además, *a priori* se conocía que tenía solución, es decir, existía un plan bien definido que podía resolverlo. En algunos casos, el sistema entraba en ciclos (iteraciones) infinitos, que detecté con "debbuging"; en otros casos empezaba a generar subplanes cada vez más grandes, sin llegar a proporcionar una solución concreta, agotando con ello la memoria de la computadora y la pila de trabajo utilizada para la recursión.

Así, un problema de planeación puede caer en cualquiera de las siguientes categorías:

1. Que el problema sea resuelto y se encuentre un plan para él.
2. Que no se encuentre una solución al problema debido a que el proceso entre en un ciclo infinito.
3. Que se agote la memoria disponible de la computadora y tampoco se logre alcanzar la meta del problema planteado.

Las categorías (2) y (3) son llamadas *anomalías* y observé que se presentan en determinados problemas cuando son especificados también de cierta forma. Sin embargo al alterar el orden de las submetas en la descripción del estado meta, el planeador era capaz ahora sí, de encontrar un plan para tales problemas.

Esta alteración del orden de la conjunción de submetas, está basada en una *jerarquización de los hechos* que se utilizan para describir las diferentes situaciones del dominio, permitiendo

con ello encontrar una conjunción de submetas equivalente que también describe al estado meta y poder encontrar un plan.

En la sección 2.3 presento un algoritmo para efectuar esta jerarquización de los hechos, la cual permite hacer los reordenamiento adecuados en las submetas que componen un meta global y de esta manera encontrar un plan para aquellos problemas que originalmente no se les podía encontrar un plan. Con esto hago más completo el sistema de planeación WARPLAN.

Considero que las ventajas obtenidas con este tratamiento son:

- Búsquedas guiadas,
- Preserva hechos protegidos,
- Planes de longitud corta,
- Un sistema más completo.

Las siguientes secciones de este capítulo proporcionan una descripción detallada sobre lo que son los sistemas de planeación, en el contexto de la inteligencia artificial. Esta descripción incluye lo que se entiende por planeación, desde un punto de vista meramente intuitivo y luego en el capítulo 2, doy su definición formal. Se encuentran también unos ejemplos sencillos de planes producidos por un planeador, e incluyo una taxonomía de las técnicas usadas por los programas que elaboran planes y una reseña histórica de su desarrollo. Esta lectura puede ser omitida por aquellos que no requieran de tal conocimiento y pueden continuar con el siguiente capítulo.

1.2 ¿ Qué es la Planeación ?

Dado que el concepto de planeación puede ser interpretado en muy diversos sentidos, voy a dar una definición sencilla, en términos cotidianos, de lo que es la planeación dentro de la Inteligencia Artificial, que es el campo que nos ocupa en este trabajo.

En el presente contexto entenderemos que **la Planeación es el proceso mediante el cual se obtiene una secuencia de acciones u operaciones, tal que si se sigue, es posible alcanzar una meta determinada a partir de una situación inicial, también dada.** A esta secuencia de acciones se le llama un *Plan.*

Por ejemplo, consideremos que nuestra meta o problema es: Pintar las paredes de un cuarto en el cual hay muebles y éstos deben permanecer en el cuarto durante el proceso de pintar y no deben ser manchados con la pintura que se usará para pintar las paredes del cuarto.

Un plan para lograr la meta anteriormente descrita sería el siguiente:

Pintar un cuarto

a. Conseguir 2 litros de pintura.

b. Conseguir una brocha.

c. Conseguir $5m^2$ de plástico para cubrir los muebles.

d. Conseguir una cubeta de plástico.

e. Preparar la pintura y colocarla en la cubeta de plástico.

f. Mover los muebles del primer lado del cuarto.

g. Cubrir los muebles del primer lado del cuarto con el plástico.

h. Colocar la escalera en la parte izquierda del primer lado del cuarto.

Tomar la brocha y la cubeta con pintura.

Subirse a la escalera.

k. Comenzar a pintar con movimientos de arriba hacia abajo y de izquierda a derecha cubriendo la superficie dentro de rango de la escalera.

Bajarse de la escalera.

m. Mover la escalera a la derecha de la posición actual

n. Repetir los pasos de (j) a (m), hasta cubrir toda la parte superior del lado que se está pintando.

o. Ir a la parte inferior izquierda del lado que se está pintando.

p. Comenzar a pintar con movimientos de arriba hacia abajo y de izquierda a derecha cubriendo toda la parte inferior del lado que se está pintando.

q Esperar media hora a que seque la pintura.

r. Colocar los muebles en su posición original.

s. Repetir los pasos de (f) a (r) para los otros tres lados restantes del cuarto.

t. Quitar el plástico que cubre los muebles.

Este plan sería producido automáticamente por un progra ma de cómputo llamado **planeador o sistema de planeación basado en conocimiento.**

De hecho, elaborar un plan es lo mismo que resolver un problema de "espacio de estados", aunque éste no es sencillo de resolver cuando el número n de elementos en el problema de planeación es grande, ya que en una representación de n-tuplos para el correspondiente

problema de espacio de estados, el número de estados posibles crece del orden de $O(n^n)$, haciendo prácticamente imposible su tratamiento con una computadora.

En muchas situaciones el planear y el hacer, casi se entremezclan en una sola cosa y no se nota una diferencia entre estas dos actividades. Por ejemplo, para resolver el "acertijo de los 8" lo que se hace en realidad, es elaborar el plan para que la computadora resuelva el acertijo. En este caso no existe una distinción clara entre la planeación y la ejecución.

Hay otros problemas en los cuales la distinción entre estas dos funciones puede llegar a ser crítica. En otros es posible saber de antemano si los pasos a dar pueden evitarse o ignorarse. En este último caso, un planeador lleva a cabo todos los pasos posibles, de la misma manera que una persona intentaría para resolver el problema. Si llegase a un punto terminal sin haber encontrado una solución, entonces se regresa hasta la última situación de decisión y toma alguna otra alternativa que haya sido desechada anteriormente. En los casos de la vida real no se puede dar marcha atrás e intentar tomar otra alternativa. En estos casos la planeación llega a tener un papel extremadamente importante, pues el tener un sistema que resuelva problemas del mundo real en donde no se puede dar marcha atrás, después de tomada una decisión, es sumamente conveniente y necesario. Aquí el sistema puede *simular* el problema del mundo real bajo distintas alternativas, esto es posible dado que se efectúa en la propia computadora. La solución encontrada mediante este proceso iterativo/recursivo de simulación puede llevarse a cabo finalmente en el mundo real.

En situaciones y aplicaciones concretas, las metas planteadas a un sistema de planeación son normalmente muy complejas y abstractas como para lograrlas en un solo paso. Es por ello que la descripción de la meta global debe dividirse en un conjunto de submetas más sencillas y simples que sean fácilmente resueltas independientemente. De esta manera, al final se combinarían los subplanes obtenidos en un solo plan completo.

En la mayoría de los casos los subplanes están conectados unos con otros, ésto es, tienen dependencias. Esto causa que en ocasiones no sea posible lograr uno de ellos sin la obtención de otro. Este aspecto es uno de los problemas más sobresalientes dentro del área de

1.3 Ejemplos de Planeación

A continuación voy a mostrar algunos ejemplos sencillos de planeación con el objeto de introducir algunos conceptos y terminología útiles en el desarrollo y comprensión de este trabajo. En el capítulo 2 (sección 2.1.5) presento la definición formal de lo que es un sistema de planeación.

El Problema de los Bloques

El primer ejemplo es el llamado en la literatura "El problema de los tres bloques" y ha sido muy utilizado por diversos investigadores para analizar diferentes aspectos de las técnicas de

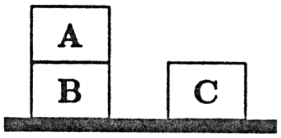


Figura 1.1: El mundo de los bloques

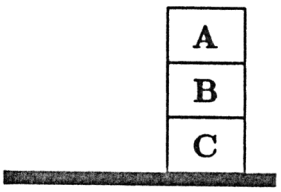


Figura 1.2: Estado Meta

la planeación [Chapman87][Manna87][Chang88], además de ser un ejemplo sencillo y comprensible.

El mundo en este ejemplo consiste de tres bloques: A, B y C y una superficie plana horizontal, que puede ser el piso o una mesa, sobre la cual están estos bloques. Los bloques a su vez pueden ser manipulados de tal forma que se pueden colocar, ya sea sobre la superficie dada o encima de otro bloque a manera de una pila, ver figura 1.1.

El tipo de problemas que se tiene en este mundo es el de colocar a los bloques en una situación determinada. Por ejemplo, poner todos los bloques sobre la superficie; formar una pila con todos los bloques, colocados éstos en un cierto orden o formar dos pilas, etc. En un mundo con más de tres bloques se podrían formar varias pilas de bloques con diferentes órdenes en sus posiciones.

Para manipular a los bloques hay una regla: Solamente se puede mover un bloque a la vez y para poder moverlo, éste no debe tener nada encima.

Una extensión de esta regla es que: Si el bloque va a ser colocado sobre otro bloque, éste último no debe tener nada encima. Y siempre es posible mover un bloque para colocarlo sobre el piso.

Un ejemplo de un problema típico en este mundo es el de obtener la secuencia de movimientos (plan), para lograr tener a los bloques como se muestra en la figura 1.2.

El plan para obtener el estado meta de la figura 1.2 a partir del estado inicial de la figura 1.1, podría ser el siguiente:

```
inicial ;
```

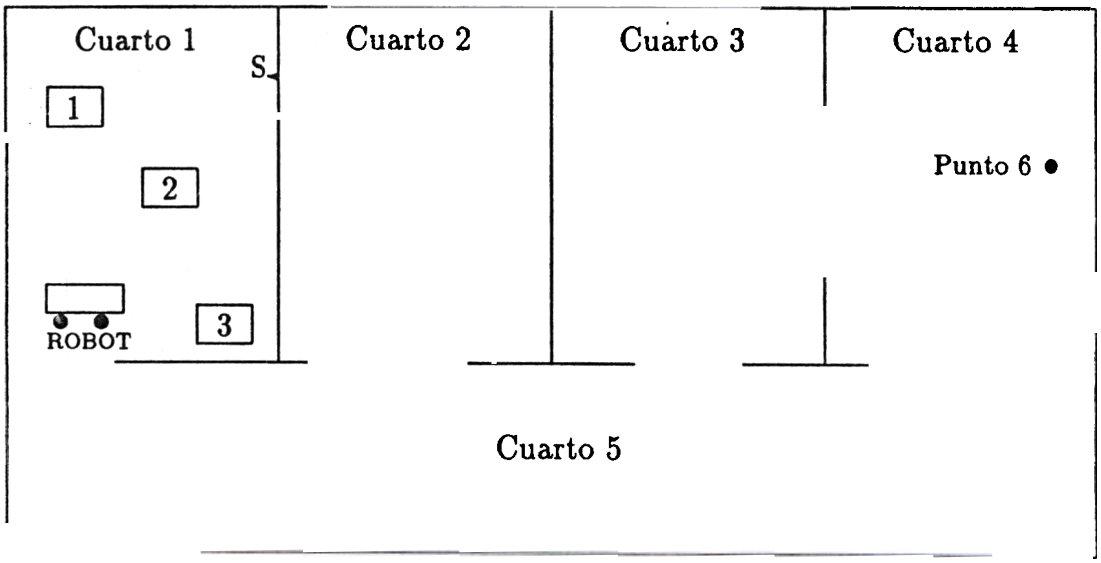


Figura .3: El mundo del problema de STRIPS

```
move(A,B,piso) ;
move(B,piso,C) ;
move(A,piso,B) .
```

"move(X,Y,Z)" representa a la acción que mueve al objeto X de la posición encima de Y a la posición encima de Z. "inicial" es el nombre del estado del cual parte el proceso.

El Mundo de STRIPS

Como un segundo ejemplo de problema de planeación voy a considerar otro de los ambientes típicos en esta área. Pertenece a un problema tipo que dió origen a la planeación dentro de la inteligencia artificial, pues el interés era el de lograr hacer de manera automática las secuencias de instrucciones que debería seguir un robot, sin la intervención humana.

El mundo de este ejemplo [Fikes 71] consiste de un ambiente cerrado en el cual hay cinco cuartos. Cuatro de ellos están conectados mediante el cuarto 5, que sirve como corredor. Ver la figura 1.3.

STRIPS deriva su nombre de "**STanford Research Institute Problem Solver**". Este planeador es el prototipo de una serie que conduce al planeador que se discute en este trabajo. En términos sencillos se puede decir que este planeador acepta la descripción mediante enunciados de un estado inicial. Y luego usando un conjunto de instrucciones o movimientos previamente definidos, elabora una secuencia de estos movimientos para lograr obtener un

estado meta, que también se describe mediante enunciados [Fikes72a].

La descripción del mundo de STRIPS es la siguiente: En el cuarto número 1 se encuentran tres cajas: Caja-1, Caja-2 y la Caja-3, localizadas en los puntos 1,2 y 3 respectivamente. Hay un robot móvil, que se puede mover libremente por todo el espacio encerrado por el ambiente y se encuentra localizado en el punto 5. Hay también en este cuarto un interruptor S en la posición 4.

Cada uno de los cuatro cuartos tiene una puerta hacia el cuarto número 5. Además en el cuarto 4 se localiza el punto 6.

El robot puede subirse y bajarse de todas las cajas. Para desplazarse, obviamente debe estar en el piso. Para encender el interruptor S debe utilizar una caja y subirse en ella. También puede empujar las cajas. Además puede transladarse por todo el medio ambiente definido en este mundo. Estas son las acciones definidas en este mundo.

Los hechos que se pueden usar en este problema son los siguientes:

HECHO	*DESCRIPCION*
	Indica que el robot debe ubicarse en el punto P
	El robot se traslada hasta donde se encuentra el objeto X
	Indica que el robot debe colocar el objeto X junto al objeto Y
	Enciende el interruptor S
	El robot se sube al objeto (bloque) B
	El robot se coloca sobre el piso
	El robot se traslada al cuarto R

Los problemas que se plantean en este mundo son tan diversos como las posibles combinaciones de los hechos anteriormente listados y en términos generales podemos decir que son todos aquellos problemas que se puedan describir utilizando los hechos anteriores, es decir, establecer tantas y diversas situaciones del mundo como se quiera.

Cualquier descripción del mundo de STRIPS puede hacerse mediante una conjunción de tales hechos. Por ejemplo para decir que se quiere que la situación final del mundo sea tal que: El interruptor S esté encendido; que todas las cajas estén juntas y que el robot se encuentre en el punto 6, podemos decirlo de la siguiente manera:

```
status (S,on) &
nextto (caja-1,caja-2) &
nextto (caja-2,caja-3) &
at(robot,6).
```

El plan para lograr tal estado del mundo sería uno como este

```
<< P L A N >>

inicial
```

```
goto2(caja-1,room(1)) ;
pushto(caja-1,switch(1),room(1))
climbon(caja-1) ;
turnon(switch(1)) ;
climboff(caja-1)) ;
pushto(caja-1,caja-2,room(1)) ;
goto2(caja-3,room(1)) ;
pushto(caja-3,caja-2,room(1)) ;
gothru(door(1),room(1),room(5)) ;
goto2(door(4),room(5)) ;
gothru(door(4),room(5),room(4)) ;
goto1(point(6),room(4)) .
```

El significado de cada una de las acciones que aparecen en el plan anterior se puede comprender sin mucha dificultad. En este ejemplo, las acciones son: goto2, goto1, pushto, climbon, climboff y gothru. Observamos que los *hechos* nos describen una situación del mundo, mientras que las *acciones* nos describen a las maneras de transformar las situaciones del mundo.

Generación Automática de Código

Como un tercer ejemplo de planeación, vamos a considerar que tenemos una máquina computadora con una arquitectura de N registros generales y un acumulador (acc). Esta máquina tiene definido además un conjunto de 6 instrucciones básicas: **suma, resta, multiplicación, división, almacenaje** y **carga** del acumulador.

El tipo de problemas que se resuelven en este medio ambiente es el de generar las secuencias de instrucciones básicas tales que al ser ejecutadas en la máquina, se obtienen estados (contenidos de los registros y acumulador) que son especificados por el usuario.

El código generado consiste de las instrucciones escritas en lenguaje ensamblador junto con un comentario en el mismo renglón que sirve para indicar el valor que tiene el acumulador después de ejecutada la instrucción. Los comentarios van precedidos por la pareja de símbolos "/*".

Vamos a considerar que las instrucciones básicas de esta máquina tienen los siguientes mnemónicos y significados:

Instrucción	*Mnemónico*	*Significado*
Suma	suma R	Suma al acumulador el valor en el registro R
Resta	sub R	Resta al acumulador el valor en el registro R
Multiplica	mul R	Multiplica el valor del acumulador por el valor del registro R
Divide	div R	Divide el valor del acumulador entre el valor del registro R
Guarda	guarda R	Almacena el valor del acumulador en el registro R
Carga	carga R	Copia el valor del registro R en el acumulador

Vamos a plantear y a resolver los siguientes problemas:

1. El acumulador tiene el valor de $(R1) + (R3)$
2. El acumulador tiene el valor de $((R2) - (R1)) * ((R3) + (R4))$
3. El acumulador tiene un valor de $(R2) * ((R3) - (R1))$ y el registro 1 tiene un valor de $(R4) - (R2))/(R1)$ y el registro 2 tiene un valor de $(R3) + (R3)/(R2))$
4. El registro 3 tiene un valor de $(R3)/((R2) - (R1))$ y el registro 1 tiene un valor de $((R2) * (R1))/((R3) - (R4))$ y el acumulador tiene un valor de $(R1) * (R4)$

En la notación del planteamiento anterior, "**(Rn)**" representa "*el contenido del registro n*".

Supongamos también que el estado inicial de la máquina es tal que los registros generales 1, 2, 3 y 4 tienen los siguientes valores:

registro 1 tiene c1
registro 2 tiene c2
registro 3 tiene c3
registro 4 tiene c4

Los planes que elaboraría el planeador para resolver cada uno de los problemas anteriores serían los siguientes:

Solución al problema (1):

```
<< P L A N >>

inicial;
carga 1    /*  c1;
suma 3     /*  c1 + c3.
```

Solución al problema (2):

```
<< P L A N >>

inicial ;
carga 3          /*  c3 ;
suma 4           /*  c3 + c4 ;
```

```
guarda _18      /*  c3 + c4 ;
carga 2         /*  c2 ;
sub 1           /*  c2 - c1 ;
mul _18         /*  (c2-c1)*(c3+c4)
```

(Nota: El registro _18 es generado por el programa)
Solución al problema (3):

```
<< P L A N >>

inicial ;
carga 4         /*  c4 ;
sub 2           /*  c4 - c2 ;
div 1           /*  (c4 - c2) / c1 ;
guarda 1        /*  (c4 - c2) / c1 ;
carga 3         /*  c3 ;
div 2           /*  c3 / c2 ;
guarda _18      /*  c3 / c2 ;
carga 3         /*  c3 ;
suma _18        /*  c3 + c3 / c2 ;
guarda 2        /*  c3 + c3 / c2 ;
carga 3         /*  c3 ;
sub 1           /*  c3 - c1 ;
guarda _20      /*  c3 - c1 ;
carga 2         /*  c2 ;
mul _20         /*  c2 * (c3 - c1).
```

(Nota: El registro _20 es generado por el programa)

Solución al problema (4)

```
<< P L A N >>

  inicial
  carga 2       /*  c2 ;
  sub 1         /*  c2 - c1
```

```
guarda _        /*  c2 - c1 ;
carga 3         /*  c3 ;
div _56         /*  c3 / (c2 - c1) ;
guarda 3        /*  c3 / (c2 - c1) ;
sub 4           /*  c3 - c4 ;
guarda _377     /*  c3 - c4 ;
mul 1           /*  c2 * c1 ;
div _377        /*  c2 * c1 / (c3 - c4)
guarda 1        /*  c2 * c1 / (c3 - c4)
carga 1         /*  c1 ;
mul 4           /*  c1 * c4.
```

(Nota: Los registros _56 y _377 son generados por el programa)

Ensamble de partes mecánicas

Como un cuarto y último ejemplo, voy a considerar uno de los problemas más interesantes en la planeación automática y es el de ensamble de partes mecánicas [Chang88].

El mundo ahora consiste de los siguientes objetos, hechos y acciones:

Objetos

Numeros:	N = {1,2,... }
Direcciones:	D = {left, right}
Ruedas:	W = {wheel 1,3,wheel N}
Ejes:	A = {axle 1,3, axle N}
Extremos:	E = {D_i end_of A_i} con $D_i \in D$ y $A_i \in A$
Orificios:	H = {hole 1, ..., hole N}
Pinza	V

Hechos:

D_1 es opuesto a D_2;	D_1 y $D_2 \in D$
W_1 unido a E_1;	$W_1 \in W$ y $E_1 \in E$
A_1 pasa por H_1;	$A_1 \in A$ y $H_1 \in H$
E_1 apunta a D_1;	$D_1 \in D$
el chasis asegurado hacia D_1;	
el chasis liberado hacia D_1;	
W_1 y A_1 se hallan sujetos;	
W_1, A_1, H_1 y V se hallan libres	

Acciones

Insertar E_1 en W_1.
Empuja W_1 de D_1 hasta D_2 por E_1 en H_1
Desliza E_1 adentro de H_1 desde D_1 .
Asegura el chasis hacia D_1 .
Libera el chasis hacia D_1 .
Sujeta W_1 .
Libera W_1 y A_1

En este mundo la pinza V se utiliza para tomar algún objeto para moverlo, sujetarlo e instalarlo. En algunos casos, es necesario colocar el chasis sobre un soporte y asegurarlo ahí para poder instalar diferentes partes en él.

Un problema típico en este mundo es el de que dado cualquier estado inicial en el que se encuentren las partes que se quieren ensamblar, se llegue a tener la configuración en donde las partes involucradas quedan ensambladas de acuerdo a una descripción determinada, por ejemplo, colocar todas las ruedas en sus posiciones correctas, lo cual se expresa de la si guiente forma:

Meta

eje A_1 pasa por orificio H_1 &
eje A_2 pasa por orificio H_2 &
rueda W_1 está unida al extremo izquierdo del eje A_1 &
rueda W_2 está unida al extremo izquierdo del eje A_2 &
rueda W_3 está unida al extremo derecho del eje A_1 &
rueda W_4 está unida al extremo izquierdo del eje A_2.

Vamos a suponer que una situación intermedia dada es la siguiente:
Las ruedas 2,3 y 4 están sin colocar. La rueda 1 está colocada en el extremo izquierdo del eje 1. Los ejes 1 y 2 pasan por los orificios 1 y 2 respectivamente y sus lados derechos están libres así como el lado izquierdo del eje 2. Y el chasis y la pinza están libres.

Para obtener la meta a partir de la descripción anterior, el planeador encuentra y proporciona el siguiente plan:

```
<< P L A N >>
 intermedio;
 asegura el chasis a la derecha;
 empuja rueda 2 de izquierda a derecha por el  lado  izquierdo
 del eje 2 en orificio 2;
 libera el chasis a la derecha;
 asegura el chasis a la izquierda;
```

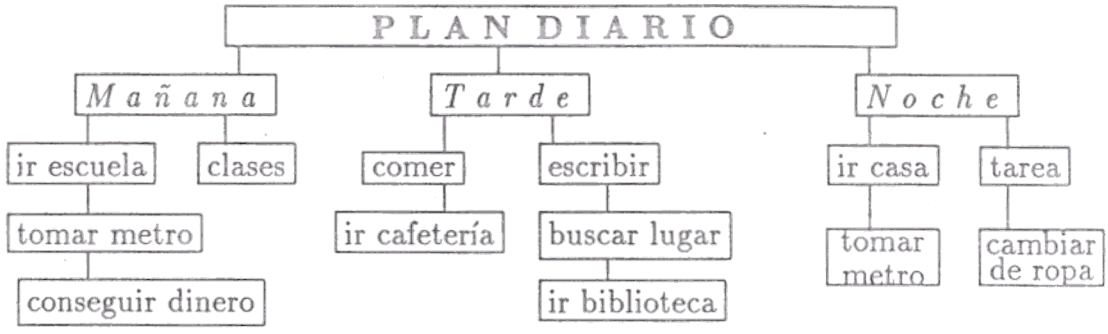


Figura 1.4 El plan diario muestra una estructura jerárquica

```
empuja rueda 3 de derecha a izquierda por el lado derecho del
eje 1 en orificio 1;
empuja rueda 4 de derecha a izquierda por el lado derecho del
eje 2 en orificio 2.
```

El conjunto de reglas y condiciones para poder llevar a cabo cada una de las acciones de este mundo es grande y por ello resulta incoveniente incluirlo en este punto. Sin embargo, el ejemplo anterior proporciona una idea clara del tipo de problemas que pueden resolver los planeadores. En el capítulo 3 se encuentra el conjunto de acciones y sus definiciones.

1.4 Taxonomía de los Sistemas de Planeación

En esta parte describiré brevemente algunos de los sistemas de planeación más distinguidos, por sus características de operación y estructuras de control, con el objeto de mostrar la ubicación del tipo de planeador que voy a discutir. También menciono a la representación del dominio y los lenguajes de programación utilizados para desarrollar sistemas de planeación.

Austin Tate escribió un artículo [Tate85] en donde proporciona una excelente lista de las técnicas utilizadas durante dos décadas de investigación sobre los sistemas de planeación basados en conocimiento, junto con ello se puede complementar una buena taxonomía de los sistemas de planeación existentes.

Hay cuatro tipos importantes de enfoques para elaborar planes: planeación no-jerárquica, planeación jerárquica, planeación basada en guión y planeación oportunística.

La mayoría de los planes tienen estructuras con las submetas anidadas,y son llamadas *estructuras jerárquicas*, como ejemplo está el que se muestra en la figura 1.4. Sin embargo la palabra jerárquico tiene otro significado bajo el contexto de sistemas de planeación, pues se usa para hacer una distinción entre los *planeadores jerárquicos* y los *no-jerárquicos.*

Así, un *planeador jerárquico* genera una *jerarquía de representaciones* de un plan en el cual la más alta representación es una simplificación o *abstracción* del plan, y la más baja consiste en un plan a detalle. Por el contrario, los *planeadores no-jerárquicos* trabajan con una sola representación para los planes. Ambos tipos de planeadores generan planes con estructuras de submetas jerárquicas como la de la figura 1.4, pero únicamente los planeadores jerárquicos usan una representación jerárquica para los planes. STRIPS es un caso típico de un planeador no- jerárquico y su versión jerárquica es ABSTRIPS [Sacerdoti73]

Obsérvese que hay tres contextos en donde usamos la palabra **jerarquía**:

Jerarquía de Representación
Estructura Jerárquica y
Planeadores Jerárquicos.

En términos sencillos podemos decir que la planeación no-jerárquica es la idea más común que se tiene de la planeación y consiste en encontrar una secuencia acciones que resuelven un problema y lograr así cada una de sus metas, (así como lo hace WARPLAN, en el ejemplo de los bloques). Este tipo de planeación puede *reducir* las metas a unas más sencillas o mediante el método de *análisis medios y fines* reducir las diferencias entre un estado actual del mundo y uno en el cual se encuentre el mundo después de alcanzar la meta final. Otros planeadores no-jerárquicos son HACKER [Sussman73], STRIPS [Fikes71] e INTERPLAN [Tate75].

La mayor desventaja de la planeación no-jerárquica es que no se puede distinguir entre las acciones que son *críticas* para el éxito de un plan de áquellas que son simplemente detalles, pues todas y cada una de las acciones definidas en un dominio dado tienen la misma importancia. En la realidad cuando alguien planea algo tiene en consideración que hay cuestiones que son importantes y otras que son muy vagas y también el nivel de detalle requerido en alguna actividad. Por ejemplo un plan muy detallado para una actividad no muy complicada es complejo en su elaboración, tal es el siguiente caso de un plan para ir a cenar: *Ir a la mesa, sentarse, tomar la servilleta, servirse un vaso de agua, tomar los cerillos, encender un cerillo y encender las velas, esperar a que sirvan la cena, tomar los alimentos.* En cambio un plan muy vago sería: *Sentarse a la mesa y comer.*

Los planes que tienen muchos detalles generan grandes espacios de búsqueda, aumentando con ello la complejidad de la solucin del problema. Y los que son muy vagos no especifican cuáles operadores deberán usarse para resolverlo. Por esto, se requiere de un equilibrio entre estos dos extremos para tener una planeación eficiente y útil.

En la *planeación jerárquica* la idea es bosquejar primero un plan completo, que es muy vago y luego refinarlo en subplanes más detallados hasta lograr un plan con una secuencia completa y detallada de operadores que resuelven el problema.

El programa ABSTRIPS es un planeador jerárquico que determina cuáles actividades son críticas para el plan e ignora todas las demás al menos al principio del proceso. Por ejemplo el problema de comprar un coche requiere que se alcancen dos submetas: *seleccionar el coche y tener el dinero* para comprarlo. Una primera aproximación del plan sería: *Seleccionar el*

coche, conseguir el dinero y comprar el coche. Este plan puede ser refinado con detalles que no son esenciales como por ejemplo: *dirigirse a donde está el coche, obtener el dinero y pagar.* ABSTRIPS elabora sus planes bajo un esquema de *jerarquía en espacios de abstracción*; en donde el más alto en jerarquía contiene un plan carente de detalles "irrelevantes" y el de más baja contiene una secuencia completa y detallada de operadores de resolución de problemas. La ventaja de este método es que al considerar primero las submetas críticas, antes que los detalles, se reduce la cantidad de búsquedas.

La planeación jerárquica fue desarrollada por primera vez por Newell y Simon en su GPS [Newell72] para demostración de teoremas en lógica, aunque ese enfoque es algo diferente al de ABSTRIPS. Otros métodos de planeación jerárquica un poco diferentes a los anteriores, los encontramos en los progamas NOAH (**N**ets **O**f **A**ction **H**ierarchies) [Sacerdoti77] y MOLGEN [Stefik81]. ABSTRIPS hace abstracción de metas críticas, mientras que NOAH la hace con los operadores de solución de problemas, inicialmente planea con operadores generales que posteriormente refina a operadores de solución de problemas de su espacio de problemas o dominio. MOLGEN va un paso más adelante, pues hace abstracción tanto de los operadores como de los objetos en el espacio de problemas.

En todos estos casos la planeación jerárquica define y planea en uno o más espacios de abstracción. Un plan se genera primero en el nivel más alto o sea en el espacio más abstracto. Esto constituye un esqueleto del cual han sido eliminados los detalles que serán considerados después en un espacio de abstracción más bajo. Este tipo de planeación ayuda a ignorar en una etapa inicial, aquellos detalles que dificultan y ocultan una solución.

En este trabajo considero una jerarquización pero *sobre los diferentes hechos* que se utilizan para describir las situaciones por las que pasa el mundo (estados), más que con los objetos u operadores de solución de problemas (*acciones*), como es el caso con los otros planeadores jerárquicos. La jerarquización sobre los hechos se dió de una manera natural al estar investigando la forma para resolver aquellas metas que hacían entrar en ciclos infinitos al planeador. Este asunto es parte de las contribuciones de mi trabajo y es detallado en las secciones que se encuentran más adelante. Por otra parte, la jeraquización de los hechos, en mi caso, es determinada en base a criterios heurísticos, aunque es posible que ello se pueda hacer mediante técnicas de *aprendizaje por ejemplos* o también con *reconocimiento de patrones* [Michalski80] para luego formar automáticamente esqueletos de planes y emplear las técnicas del tercer enfoque de la planeación que se describe en el siguiente párrafo. El uso de estas dos técnicas es un tema de investigación que está abierto.

El tercer enfoque para elaborar planes, llamado *por esquemas* ("scripts"), también está basado en el uso de planes-esqueleto, pero a diferencia de la planeación jerárquica estos esqueleto son tomados de un almacén en lugar de ser generados. Este enfoque fue utilizado en una de las versiones de MOLGEN [Stefik81]. Los planes almacenados contienen los esquemas para resolver diferentes clases de problemas. Abarcan desde planes extremadamente específicos para problemas comunes hasta planes muy generales para una amplia gama de problemas. La técnica consiste en encontrar primero un plan-esqueleto en el almacén que se pueda aplicar para resolver el problema dado, luego los pasos abstractos del plan son llena-

dos con los operadores de solución de problemas para el contexto actual del problema. Este proceso de instanciación involucra gran cantidad de conocimiento del dominio específico. Frecuentemente se procesa a través de varios niveles de generalidad hasta que se encuentra un operador de solución de problemas que cumple con cada uno de los pasos del plan-esqueleto.

El cuarto enfoque a la creación de planes es llamado *oportunístico* y es debido a Hayes-Roth [Hayes-Roth80]. Este método se caracteriza porque tiene una estrategia de control más flexible que los otros enfoques. Este método maneja un esquema control de *pizarrón* para modelar la planeación humana [Hayes-Roth78]. El pizarrón funciona a manera de "agencia de información" que hace sugerencias, alimentado por un conjunto de especialistas, acerca de los pasos para lograr los planes. Cada *especialista* hace decisiones particulares en la planeación. Los especialistas no operan en un orden particular, la falta de sincronía en las decisiones de la planeación se hace cuando hay una razón para hacerlo, de aquí se deriva el nombre de *oportunística.*

En este enfoque se incluye una componente *hacia-arriba* ("bottom-up"), pues se dirige mediante oportunidades que incluyen acciones de detalle en la solución de problemas. Al contrario de lo que pasa en la planeación jerárquica en donde el proceso de refinamiento es *hacia-abajo* ("top-down"), en el cual las acciones de detalles no se deciden sino hasta el último momento en el desarrollo de un plan.. También se puede distinguir otra diferencia de este método con los otros y es que puede hacer *islas*[Sacerdoti74] de acciones de planeación (partes de un plan) independientes, lo que no sucede con los otros planeadores que tratan de desarrollar un plan monolítico para cada nivel de abstración.

Metas Conjuntivas.

La planeación, como la hemos definido, informalmente, consiste en encontrar una secuencia de acciones que permitan pasar de una situación inicial dada a una situación final o meta, también dada.

La cuestión es ¿Cómo especificar tales situaciones de un problema concreto? Bueno, la respuesta aparece cuando nos preguntamos Qué son las situaciones en el mundo o dominio dado? Consideremos en este momento una idea intuitiva de lo que es un *hecho* diciendo que es una *relación entre los objetos* del mundo. Y vamos a decir que una *situación* determinada del mundo, en un momento determinado es la descripción del mundo mediante todos y cada uno de los hechos que están interviniendo en ese momento. Para interpretar esto matemáticamente decimos que *una situación es la unión de todos los hechos que describen al mundo en un momento dado.*

Es por esto que la especificación de la situación final o meta es la conjunción de las submetas que la componen (hechos). En la literatura a esta forma de representación de los estados finales o metas se le llama "**metas conjuntivas**".

Linealidad en los Sistemas de Planeación.

(a) (b)

Figura 1.5: Estado inicial Estado meta

Los planeadores se pueden dividir en dos tipos básicos de acuerdo a si usan o no una estrategia para resolver un problema de planeación especificado con metas conjuntivas. Esta estrategia es llamada **"Suposición de linealidad"** y divide a los planeadores en *lineales y no-lineales* de acuerdo a si la usan o no.

Esta "linealidad" en la planeación nos dice que los subplanes generados para cada una de las submetas (hechos) de una meta conjuntiva están "*desacoplados*".

Decimos que *dos subplanes están desacoplados* si al resolver y llevar a cabo uno de ellos completamente, sus acciones no requieren de la ejecución de alguna de las acciones del otro subplan.

Para algunos problemas esta interacción entre las acciones de dos subplanes es necesaria. Cuando esto ocurre se dice que *el plan es no-lineal* (pues no está formado por una secuencia lineal de subplanes completos) Los planeadores que hacen ésto son llamados no-lineales y su estrategia para resolver el problema es la que usa el enfoque de "mínimo compromiso", representando un plan como una red parcialmente ordenada (un grafo) de acciones y metas e introduciendo los eslabones ordenados entre las acciones y las metas cuando la solución así lo requiera.

Este último tipo de planeadores son los más usados, pues muchos problemas se pueden resolver únicamente con esta estrategia. Por ejemplo la "anomalía de Sussman" sólo se puede resolver con el enfoque de la no-linealidad, veamos porqué.

Esta anomalía consistía en que si al resolver una submeta, se destruía una submeta anteriormente lograda, entonces se hacía un intercambio del orden en el cual se resolvían las submetas y ahora se resolvía primero la submeta cuya solución interfería con la ya lograda, y luego se resolvía ésta. Si al resolver la segunda submeta también destruía la submeta anterior ya lograda, entonces el sistema declaraba insoluble el problema, aunque un sujeto humano sí pudiera resolverlo.

Consideremos, por ejemplo, que tenemos el estado inicial que se muestra en la figura 1.5 a y queremos alcanzar la situación final mostrada en la figura 1.5 b.

Si para resolver el problema se inicia por la submeta **sobre(A,B)** habría que quitar a C y dejar libre a A. Luego colocar A sobre B. Pero luego al querer lograr la segunda submeta **sobre(B,C)** se tendría que destruir el subplan ya logrado anteriormente. Si por el contrario comenzamos por resolver primero **sobre(B,C)**, lo primero sería pasar el bloque B del piso

a encima de C y se tendría el primer subplan. Pero pasa lo mismo que en el caso anterior al querer ahora lograr la otra submeta: **sobre(A,B)**. Como se ve, si seguimos una estrategia de suposición lineal no encontramos una solución para este sencillo problema.

Un enfoque no-lineal de este problema primero resuelve **sobre(A,B)** quitando a C de A y colocándolo sobre el piso. Y ahora, sin terminar este subplan, va a lograr la siguiente submeta, **sobre(B,C)**, poniendo a B sobre C. Ya logrado este subplan, se regresa al que dejó pendiente, terminándolo colocando a A sobre B.

Planeación No-lineal de metas conjuntivas.

El problema de la planeación con metas conjuntivas (planeación conjuntiva) ha sido un tema de investigación de los sistemas de planeación durante los últimos 10 años.

El problema o la idea principal consiste en lograr varias metas simultáneamente, es decir, encontrar un plan que hace a una fórmula conjuntiva verdadera, después de que ha sido ejecutada.

Un planeador para ser útil debe ser independiente del dominio. Y Chapman después de haberlos analizado ha llegado a la conclusión de que todos los planeadores de este tipo, funcionan de la misma manera [Chapman87].

La dificultad en la planeación conjuntiva independiente del dominio está en la interacción entre los medios para lograr las metas individuales.

Un problema clásico que muestra esta dificultad es "la anomalía de Sussman", que se menciona en la siguiente sección.

Del árbol genealógico de los sistemas de planeación, los planeadores no-lineales, conjuntivos e independientes del dominio, han sido muy utilizados y ofrecen buenas expectativas de aplicación inmediata a problemas concretos. En este trabajo discutiré la representación formal de lo que es un plan a nivel general de esta clase de planeadores. También presentaré las ideas sobre algunos problemas involucrados con ellos y una técnica desarrollada en este trabajo de investigación para dar algunas soluciones.

Lenguajes de programación.

Los primeros sistemas de planeación fueron escritos en lenguajes diseñados exprofeso, ejemplo de ello fueron PLANNER [Hewitt72], POPLER [Davis73], CONNIVER [Sussman72] y sus derivados. La estrategia de búsqueda en profundidad es utilizada a través del lenguaje PROLOG [Clocksin84] para construir WARPLAN [Warren74], que es probablemente el primer planeador escrito en este lenguaje. En muchos otros casos fueron utilizados los lenguajes típicos en inteligencia artificial, QA4 [Rulifson72], QLISP [Sacerdoti76], así como LISP y casi todos sus dialectos. En algunos otros casos, los sistemas se han implementado como una combinación de un lenguaje de IA y un lenguaje para representar conocimiento tal como UNITS [Stefik79] escrito en LISP (para MOLGEN [Stefik81]), HBASE [Barrow] en POP-2/POPLOG [Sloman83] (para INTERPLAN [Tate75] y NONLIN [Tate77]), etc. Por otra

parte se sigue investigando en la representación del conocimiento mediante lenguajes apropiados para ello, tales como KRL [Bobrow76], PEARL [Deering81], LOOPS [Bobrow83], KEE, SRL [Fox83]. Knowledge Craft, ART, etc. En lo que se refiere a computación y máquinas en paralelo, se encuentran NETL [Falhman79], FACT [McGregor77] y la máquina de heterarquía reconfigurable [Guzmán84].

1.5 Reseña de los planeadores

Los antecedentes de los sistemas de planeación se remontan al principio de los años 60's. Las primeras técnicas fueron introducidas por el sistema GPS (*General Problem Solver*) [Newell63], en particular la técnica llamada "análisis medios y fines" [Tate85] que consiste en aplicar un operador, seleccionado entre varios operadores posibles, que logre reducir cierta diferencia entre la descripción del estado actual y la descripción del estado meta del problema original.

Otro de los primeros planeadores fue **STRIPS** [Fikes71], que es un resolvedor de problemas para robots. En este modelo se introduce por primera vez la idea de *las postcondiciones*, que son los hechos que se garantizan que existen o son verdaderos después de que ha sido ejecutada una acción. Con esta idea se tiene un *Axioma de Marco*, que establece que todos los hechos que determinan una cierta situación del mundo prevalecen después de que una acción ha tenido efecto en el mundo, con excepción de aquellos que son eliminados explícitamente por la ejecución de la acción, y por otro lado son anexados otros nuevos hechos debido también al efecto de la acción sobre el mundo. Todo ello define la nueva situación del mundo.

Las técnicas introducidas por STRIPS son claramente definidas y manejadas en el sistema HACKER [Sussman73] mediante listas de hechos llamadas ADD y DELETE. Cada una de ellas indica al sistema cuáles hechos son añadidos y cuáles eliminados de la descripción de una situación dada del mundo sobre la cual tiene actuación una acción determinada. En este sistema se detectó un problema conocido como "la anomalía de Sussman", que fue resuelto en la tesis doctoral del mismo autor, Gerald J. Sussman.

Este problema fue eliminado al siguiente año (1974) en los sistemas WARPLAN de David H. Warren [Warren74] y en INTERPLAN de Austin Tate [Tate74][Tate75], utilizando dos métodos para lograr submetas: *por extensión y por inserción*. El método por extensión va logrando una a una las submetas de una meta final. Si encuentra que alguna submeta destruye algo de lo ya logrado, entonces entra a funcionar el método por inserción, el cual hace un "backtraking" sobre el plan parcial, que se lleva logrado hasta el momento y trata de localizar un punto dentro de éste, para insertar la solución de la submeta presente. Es claro que la inserción de esta solución dentro del plan parcial no destruye lo que sigue dentro él.

El primer planeador no-lineal fue desarrollado en 1975 por Earl D. Sacerdoti y fue llamado NOAH [Sacerdoti77]. Con este planeador se introdujeron algunas mejoras con respecto a la representación de los planes (los planes se representaban como listas de acciones parcial y no

completamente ordenadas), aumentando también el conjunto de operaciones de modificación de planes. Posteriormente Austin Tate hizo algunas mejoras a NOAH dando como resultado un nuevo sistema de planeación llamado NONLIN [Tate76][Tate77], que también aumentó el conjunto de operaciones de modificación de planes.

El sistema MOLGEN [Stefik81a] fue otro de los sistemas de planeación no-lineal y su mérito estuvo en que fue un sistema que introdujo a las restricciones como una técnica de la planeación.

Posteriormente, todos los sistemas de planeación desarrollados siguieron el modelo de NOAH, con algunas modificaciones adicionales para manejar la representación del tiempo.

Se puede considerar que el área sobre planeación ha sido investigada con muy pocos logros durante las dos últimas décadas y se puede decir que existen muchos temas de investigación abiertos, a pesar de que ya en 1972 se tenía una situación muy parecida a la actual [Fikes72b].

Actualmente la planeación está siendo considerada por algunos investigadores como una aplicación de la Deducción Automática [Manna87]. Para presentar este novedoso enfoque se ha desarrollado una variante de la lógica situacional [McCarthy63], llamada *Teoría de Planes*, en la cual los planes son objetos explícitos.

Los autores de este tratamiento han trabajado varios años en la síntesis de programas (derivación automática de un programa de cómputo que logre una especificación dada). Esto se apoya en un enfoque deductivo en el cual la derivación de un programa se considera como una tarea en la prueba de teoremas [Manna80] [Manna85].

La razón de esta variante se debe a que la lógica situacional convencional tiene algunas deficiencias cuando es aplicada al problema de planeación y con esta "teoría de planes" se evitan esas deficiencias.

La deficiencia primordial es que mediante la lógica situacional es posible encontrar planes que no sean ejecutables. Sin embargo con la teoría de planes se puede demostrar a priori que no hay solución para determinados problemas planteados.

De hecho, una variante de esta lógica fue introducida en el sistema de planeación QA3 [Green69], en donde los símbolos para las funciones y predicados, cuyos valores cambian, se les asignó argumentos relacionados con los estados, esto es, los estados se manejan como argumentos de las propias funciones y predicados. Igualmente, las acciones tenían dentro de sus argumentos al estado sobre el cual se estaba actuando.

Para construir un programa, se prueba un teorema que establece la existencia de una salida que da las condiciones especificadas. La prueba se restringe a ser constructiva, esto es, en que se debe describir un método de cómputo para encontrar la salida. Este método es la base del programa que se extrae de la prueba.

Los planes se pueden ver como programas imperativos, en donde las acciones corresponden a las instrucciones para la computadora, las pruebas a los condicionales del programa y el dominio o mundo a la estructura de datos.

Esta analogía sugiere que la técnica para la síntesis de programas imperativos puede llevarse también a cabo en el dominio de la planeación.

Bajo el enfoque de la teoría de planes las partes más difíciles e importantes del método

son las que están involucradas en probar que se alcanzan satisfactoriamente las condiciones especificadas inicialmente (metas), más que en generar el propio plan.

Un planeador como HACKER [Sussman73] puede encontrar mediante correcciones sucesivas las soluciones que se encuentran con la teoría de planes pero a diferencia de ésta, no puede demostrar si el plan encontrado es correcto o no. Su confianza descansa más bien en el conocimiento de alto nivel que tiene alimentado en su base de conocimiento.

La inferencia imprecisa puede ser necesaria para las aplicaciones de la planeación, pero para la síntesis de programas convencional es más adecuada la demostración formal de teoremas. Algunos creen que el enfoque basado en la demostración de teoremas ("theorem proving") es inadecuado como técnica de planeación pues —argumentan—, "un demostrador de teoremas de próposito general nunca podrá competir contra un sistema cuyas estrategias están diseñadas especialmente para solución de problemas" ("problem solving").

Lo anterior es motivado en razón de que cuando se trabaja con las técnicas de "Demostración de Teoremas" (**DT**), los métodos y técnicas son más serios y más sólidos, desde un punto de vista formal, y por lo tanto no hay duda sobre los resultados obtenidos cuando son aplicados áquellos. El problema con estos métodos es que sus espacios de búsqueda llegan a ser muy complejos y las soluciones muy largas.

En cambio cuando se utiliza un enfoque de "Solución de Problemas" (**RP**), las búsquedas son sencillas, pero queda abierto el problema sobre si lo obtenido es una solución. Normalmente se considera que el lograr alcanzar una meta garantiza que el plan (trayectoria) es correcto.

Manna y Waldinger [Manna 87] proponen la unión intercalada de un demostrador de teoremas de propósito general con un componente especial que tenga las técnicas específicas de planeación, para llegar a tener un sistema de planeación mejor que los sistemas de planeación convencionales. De hecho el sistema WARPLAN [Warren74] que he utilizado en este trabajo se puede considerar como un *probador de teoremas de la lógica situacional equipado con una estrategía de planeación similar a la de STRIPS* (RP)[Fikes71].

Es por esto último que en mi trabajo de investigación seguí con la lógica situacional convencional y considero que la lógica de la teoría de planes de Manna y Waldinger junto con los aspectos de consistencia y completitud de los sistemas de planeación, que establezco en este trabajo, conducen a desarrollar un nuevo trabajo de investigación que está abierto, el cual conduciría a tener una mejor base teórica para desarrollar planeadores más confiables. Esto es, que sean *completos* en el sentido de que si un problema tiene solución, la encuentra, y que sean *sólidos*, es decir, que las soluciones encontradas sean correctas.

Finalmente, uno de los últimos trabajos sobre sistemas de planeación es debido a David Chapman, quien desarrolló e implementó un algoritmo, TWEAK [Chapman87], que combina y destila el estado del arte sobre técnicas de planeación.

Chapman analiza los trabajos anteriores al suyo sobre la planeación de metas conjuntivas independientes del dominio y llega a la conclusión de que todos los **planeadores conjuntivos** lineales y no-lineales funcionan de la misma manera y que la eficiencia y el funcionamiento correcto de estos planeadores dependen de la representación tradicional para las

acciones, mediante listas "add/delete", las cuales según él, reducen drásticamente su utilidad. Sin embargo, presenta unos teoremas que *sugieren* que es imposible tener un planeador de propósito general que sea eficiente aún con representaciones más expresivas para las acciones.

TWEAK es una reconstrucción matemática rigurosa de todos los planeadores no-lineales anteriores a él. Y además es un algorítmo implementado en un programa de computadora. Está basado en un teorema que establece lo siguiente: "Si TWEAK, dado un problema, termina dando una solución, entonces el plan que produce resuelve el problema. Si TWEAK regresa un mensaje de fallo o no se para, entonces no hay una solución para el problema propuesto".

El programa no tiene detección de ciclos infinitos y Chapman sugiere utilizar técnicas heurísticas o pruebas para detectar problemas sin solución. La pregunta importante es cuándo la planeación es un problema decidible o si es posible hacer un planeador que siempre se pare. Esto lo condujo a su primer teorema de indecidibilidad, que dice: "Cualquier máquina de Turing con su entrada, puede ser codificada como un problema de planeación en la representación de TWEAK. Por lo tanto, la planeación es indecidible, y la cantidad de tiempo requerida para encontrar una solución no está acotada superiormente".

Según el propio Chapman, este teorema tiene dos puntos débiles:

1 Que para su demostración usa las cintas infinitas de las máquinas de Turing y sucede normalmente que tenemos problemas de planeación con un número finito de estados, aunque no siempre es garantía de que ello implique un espacio de búsqueda finito.

2. Que la demostración del teorema muestra que el tamaño del plan más corto para resolver un problema puede ser arbitrariamente grande, en lugar de que el proceso de planeación mismo sea complejo.

De todo lo anterior podemos ver que existen dos enfoques principales para resolver los problemas de planeación. Uno es mediante el enfoque de **DT** con el cual se establece una lógica formal y un lenguaje de primer orden con sus respectivas reglas de inferencia. Sus métodos de demostración tienen las siguientes características:

- Hacen búsquedas exhaustivas
- Son completos y sólidos
- Son poco eficientes en cuanto a la longitud de sus planes

Los planeadores que utilizan este enfoque aparecen históricamente con la llamada lógica situacional (NOAH, MOLGEN) y luego la "Teoría de Planes" de Manna y Waldinger.

Son completos en el sentido de que si hay una solución para un problema, la encuentra. Y sólido que la solución encontrada es correcta.

El otro enfoque es el de RP con el cual los planeadores encuentran planes paso a paso, de manera directa, su búsqueda es guiada y se van protegiendo los hechos logrados. Este enfoque es utilizado en por los primeros planeadores, por ser los sucesores directos de los resolvedores de problemas. Uno de estos planeadores, de los típicos, es STRIPS. Aquí se detectó la llamada anomalía de Sussmann, la cual posterioremente quedó resuelta. Uno de los sistemas que la resolvieron fue WARPLAN, que es la siguiente generación de los planeadores.

Utilizando este enfoque se mantiene cierta incompletitud en los sistemas de planeación.

Chapman con su TWEAK sigue el enfoque de DT y por lo tanto TWEAK resulta ser completo, pero poco eficiente, no pudiéndose mejorar por no poder detectar ciclos. Y por lo tanto no poder reconocer en general, si existe o no un plan para una meta planteada en un sistema de planeación dado. En este sentido TWEAK es completo pero indecidible.

Además no es finito y tampoco es determinístico y en términos generales la codificación de cualquier sistema de planeación es muy compleja e irrealizable.

Debido a la representación tan sencilla que usa para sus planes lo hace tan restrictivo que no se pueden representar la mayoría de los dominios, pero sin estas restricciones TWEAK no sería completo y tal vez no daría soluciones correctas. Es por las restricciones sobre la representación de las acciones, que TWEAK es casi inútil como planeador del mundo real.

Considero que con mi trabajo rescato el enfoque de RP para los planeadores y los hago más completos con las técnicas de DT. Logrando con ello el poder resolver una gran cantidad de problemas de planeación del mundo real y teniendo un grado significativo de completitud en el sistema. El cual es finito, determinístico, eficiente y cuenta con un mecanismo de demostración de correctitud de los planes que encuentro.

TWEAK se hace completo precisamente porque pierde su determinismo. En mi caso, si quiero salvar el determinismo hay que sacrificar algunas propiedades de completitud. Esto se obtiene analizando los estados (análisis "situacional") para obtener una forma de cómo dirigir heurísticamente las búsquedas.

Esta guía heurística se basa en un reordenamiento de las submetas que componen el estado final o meta, de tal manera que "*la obtención de una submeta no interfiera con las submetas posteriores*".

Capítulo 2

TEORÍA DE LA PLANEACIÓN

2.1 Formalización de los Sistemas de Planeación

En este capítulo presento las contribuciones principales de este trabajo de investigación. El primer objetivo es el de establecer las bases matemáticas de los sistemas de planeación, para poder verlos como teorías lógicas. En ello se apoya la discusión del segundo objetivo, que es el de analizar los aspectos de la consistencia y completitud de los sistemas de planeación, considerados éstos como sistemas formales. Este tema tiene una fuerte importancia en la solución de las anomalías planteadas en el capítulo anterior.

Y el tercer objetivo es el de presentar y estudiar una función matemática cuyo uso permite definir una jerarquía para los hechos involucrados en las descripciones de los estados. Con esta jerarquización es posible encontrar trayectorias de soluciones entre los estados inicial y meta de aquellos problemas que, sin este proceso de jerarquización, presentan dos tipos de anomalías:

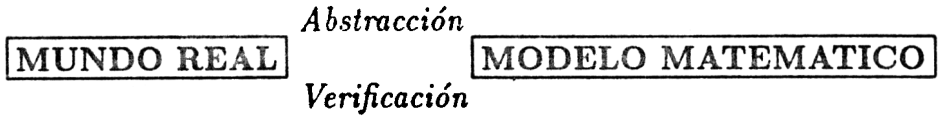
- Entrada a un ciclo infinito.

- Agotamiento del espacio en memoria antes de encontrar una solución

Para manejar matemáticamente los diversos aspectos involucrados con la planeación, es conveniente tener claro lo siguiente:

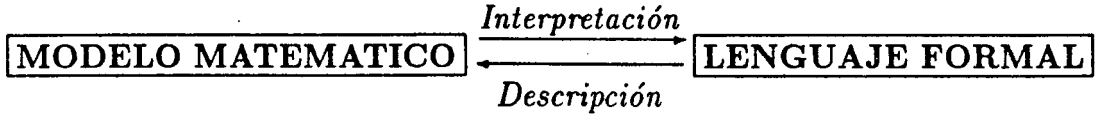
Del mundo real, queremos ubicarnos solamente en una parte del mismo, sobre la cual nos interesa resolver una serie de problemas que de antemano nos hemos planteado.

Para resolverlos, una manera de hacerlo, es mediante un modelo matemático, el cual nos permita representar lo mejor posible a aquella parte del mundo que nos interesa y describir los problemas y sus soluciones. La construcción es tal que las soluciones encontradas mediante el modelo correspondan también a una solución en el mundo real:



Este modelo es una abstracción del mundo real y para poder trabajar con él, utilizamos entidades ideales, que son manejables con técnicas matemáticas como, digamos, la teoría de conjuntos. En este capítulo construiremos un modelo consistente de entes como cadenas de átomos, listas, conjuntos, relaciones, mapeos, etc.

Para hablar del modelo y de lo que sucede en él, se requiere de un lenguaje formal que nos permita describirlo y luego poder hacer algunas interpretaciones de las transformaciones simbólicas que se hagan con él y ver que reflejan lo más relevante del modelo.



Componiendo las flechas mostradas en las figuras anteriores, podemos pasar del lenguaje al mundo real.

En nuestro caso, tenemos que el problema fundamental de la planeación es *encontrar planes* en diversos ámbitos del mundo real. Algunos ejemplos ya los he dado en el primer capítulo.

Para esto, vamos a considerar un esquema de un modelo matemático que nos permita tratarlo y resolverlo de una manera lo más general posible.

2.1.1 Motivación

Cualquier modelo debe representar una parte del mundo real, en la cual se puede identificar la presencia de diversos objetos. Normalmente hay un interés en determinadas relaciones que guardan estos objetos entre ellos, para un momento dado. En cada momento la configuracipn de tales relaciones nos determinan un estado en el modelo. En nuestro caso los estados pueden cambiar, debido a la acción de agentes que actúan sobre ellos. Esto hace del modelo un sistema de transicipn entre estados. Por ello se requiere, también, de un sistema formal que nos permita manejar esas transiciones.

Para describir este modelo debe utilizarse un lenguaje que conste de los siguientes elementos:

- Un alfabeto de símbolos para variables, constantes, relaciones y acciones.

- Elementos compuestos, los cuales estsn definidos en el alfabeto y han de describir y manejar las situaciones de los objetos.

- Y que podamos describir a los agentes que actúan sobre los objetos y la manera de cómo lo hacen.

Aunque la manera más natural de construcción es pasar del mundo real al modelo y luego de éste a la definición del lenguaje, optaré por conveniencia y claridad en la presentación, por construir primero el lenguaje y luego a partir de él, construir el modelo particular.

Posteriormente veremos la interpretación del lenguaje en *ese* modelo. Y luego verificaremos que el modelo así construído modela en efecto el mundo real.

Los sistemas de planeación han de implementarse en una computadora. Para ello utilizamos el lenguaje formal, y debemos estar seguros que los resultados de la computadora en ese lenguaje efectivamente corresponden con la modelizacion del problema y a su vez con lo que se quiere resolver en el mundo real. Tenemos pues que:

- Los problemas de planeación que se presentan en el mundo real cuando son tratados de manera automática, son reducidos, de alguna manera a símbolos que son reconocidos en una computadora por medio de un programa de cómputo, en este caso un *planeador.* La manipulación simbólica de estos problemas sigue sus propias reglas sintácticas, las cuales, aunque correctas desde el punto de vista matemático, en algunos casos pueden representar a situaciones irreales, tal es el caso cuando formulas bien formadas describen situaciones que no existen en el mundo real. Si pensamos en el mundo de los bloques, tenemos que la fórmula on(X,X) es correcta sintácticamente pero no es posible realmente.

- El modelo proporciona información parcial y aproximada del mundo real.

2.1.2 El Lenguaje

El lenguaje va a constar de un *ALFABETO* cuyos *elementos primitivos* son los siguientes

a *Símbolos de Objetos.* Son símbolos que se utilizan para nombrar a los diferentes objetos del dominio. Podemos considerarlos como las constantes del lenguaje.

$$Objeto \mapsto Símbolo$$

b *Variables.* Son símbolos que no nombran a un objeto en particular, sino que pueden representar a cualquier objeto al que no podamos nombrar en un momento dado. Es decir, tiene como rango el conjunto de objetos.

c *Símbolos de Relaciones.* Corresponden a los nombres de las relaciones entre los objetos. Una *relación entre objetos*, se representa sintácticamente mediante un **identificador**, que es *el nombre de la relación.* Seguido de una lista de símbolos de objetos o también de variables:

$$identificador < (\ símbolos \quad variables)^* >$$

(Nota: El asterisco significa: "*...tantas veces como se quiera, inclusive ninguna*")

El *número de símbolos* (y/o variables) puede ser cero o cualquier número. Y a este número se le llama la *ARIDAD* de la relación. Por ejemplo, las relaciones **sobre(A,B)** y **nextto(R,A)** tienen aridad=2. La relación **onfloor** tiene aridad=0.

Los *ELEMENTOS COMPUESTOS* del lenguaje son los siguientes:

a *Atomos.* Si *rel* es el nombre de una relación cuya aridad es n, y $x_1, \ldots, x_n$ representan objetos o variables, entonces al instanciar $rel(x_1, \ldots, x_n)$ obtenemos un *ATOMO.*

En términos generales vamos a decir que cualquier instancia de una relación dada, es un *átomo* y lo representamos como:

$$\text{átomo ::=< nombre\_relación > ([argumentos]}^{\dagger})$$

A un átomo lo consideraremos también como un predicado, esto es, que puede ser verdadero o falso.

b *Hechos.* Un hecho es un átomo cuyos argumentos son todos símbolos de objetos, esto es, *son átomos que no tienen variables.*

c *Conjunción de átomos y frases.* Formalmente la conjunción lógica de átomos está definida de la siguiente manera:

$$conj.de\acute{a}tomos = \acute{a}tomo_1 \wedge . \quad \wedge \acute{a}tomo_n = \bigwedge_{i=1}^{n} \acute{a}tomo_i$$

Una conjunción de átomos será verdadera, si todos los átomos que la componen lo son.

Además vamos a convenir que *la conjunción vacía es verdadera* en cualquier estado.

A las conjunciones de átomos las representaremos por *frases*, las cuales son listas de átomos:

$$(a_1, \ldots, a_n)$$

La representación es natural, una frase representa a la conjunción de sus componentes.

Hay dos operaciones, $\oplus$ y $\ominus$, sobre las frases, las que defino a continuación:

Definición 1 *Si* $A = (a_1, \ldots, a_n)$ *y* $B = (b_1, \ldots, b_m)$ *son dos frases, tales que* $m \leq n$.

Entonces la operación $\oplus$ *entre* A *y* B, *es otra frase que está definida como:*

$$A \oplus B = (a_1, \ldots, a_n, b_1, \ldots, b_m)$$

es decir, $A \oplus B$ *es la concatenación de* A *y* B

ii *Y,* $A \ominus B$ *es la frase formada por los átomos que quedan de la frase* A, *después de que han sido removidos los átomos iguales que aparecen en* B.

Por ejemplo si $A = (a, b, c, d)$ y $B = (e, f, b)$, entonces: $A \ominus B = (a, b, c, d, e, f, b)$ y $A \ominus B = (a, c, d)$.

La *longitud* de una frase es el número de átomos que tiene la frase. Por ejemplo, la longitud de (a, b, c) es 3, la de (a) es 1, la de () es 0, etc.

Una *SUBFRASE* de alguna frase F es una frase tal que todos sus átomos también son átomos de F.

d *Oraciones.* Las oraciones son frases cuyos átomos van a estar todos instanciados, es decir, son frases formadas con hechos. Por eso tenemos que: $\{oraciones\} \subset \{frases\}$.

EQUIVALENCIA DE FRASES.

Veamos ahora el concepto de *Equivalencia* dentro del conjunto de frases. Para ello introducimos tres reglas de operación para listas:

Regla 1: *Conmutación*
"Sustituya una pareja *ab* por la pareja *ba*."
(en donde *a* y *b* son átomos).

Regla 2: *Idempotencia*
"Sustituya una pareja *aa* por el átomo *a*.

Regla 3: *Redundancia*
"Sustituya a un átomo *a* por la pareja *aa*."

Definición 2 *Dos frases son equivalentes si se pasa de una a otra mediante la aplicación de un número finito de veces de las reglas de conmutación, idempotencia y/o redundancia.*

La relación definida entre frases es, en efecto, de equivalencia, es decir, es reflexiva, simétrica y transitiva. Produce pues, *clases de equivalencia* en el conjunto de las frases. La definición de esta relación de equivalencia claramente está motivada por representación que hemos hecho de conjunciones de átomos como frases.

Dentro de cada clase de equivalencia hay frases que tienen una longitud mínima. A estas frases se les llama *Frases Canónicas.*

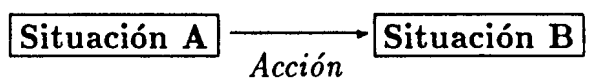
Es fácil demostrar que para cualesquiera dos frases canónicas, una se obtiene mediante una permutación de la otra. Aquí cabe hacer notar que una permutación se puede ver como una aplicación sucesiva de la regla de conmutación.

ACCIONES.

Al definir de manera informal la Planeación, se mencionaron también a las *secuencias de acciones* o movimientos para pasar de un estado o situación a otra.

En el lenguaje matemático utilizado para interpretar al modelo, las acciones actúan sobre los objetos y se consideran como elementos adicionales a los primitivos.

Hasta este punto hemos considerado, intuitivamente, que dado un estado o situación del mundo, si le aplicamos una acción, se modificará áquel produciendo con ello una nueva situación.



Vistas las acciones dentro del lenguaje, representan reglas de producción que actúan sobre las oraciones, transformando una oración en otra oración:

$$\{oraciones\} \overset{\text{ACCION}}{\longrightarrow} \{oraciones\}$$

$$\alpha(\mathcal{O}_1) = \mathcal{O}_2$$

en donde α representa a una acción y $\mathcal{O}_1$ y $\mathcal{O}_2$ son oraciones.

Sintácticamente una acción se representa mediante un nombre seguido de una lista de argumentos que pueden ser tanto objetos como variables. Y se representa simbólicamente como sigue:

nombre_acción (lista_de_argumentos)

Por ejemplo, en el caso del problema de los tres bloques, la acción definida en ese mundo es "*move(A,P,Q)* ", que representa a la acción de mover el bloque A de la posición que tiene sobre P a una posición nueva sobre Q.

Cada acción α tiene asociadas tres funciones: *puedese, añade* y *borra*, definidas las tres sobre el conjunto de las acciones al conjunto de frases:

$$puedese : acciones \rightarrow \{frases\}$$
$$añade : acciones \rightarrow \{frases\}$$
$$borra : acciones \rightarrow \{frases\}$$

La función *puedese* asocia a cada acción una frase. Esta frase indica condiciones necesarias para que la acción se pueda aplicar a una oración.

La idea de las dos funciones, *añade* y *borra*, es que la primera nos indicará los átomos que deberán ser anexados a una oración, después de que ha sido aplicada la acción. Y la segunda nos dirá cuáles átomos deben ser removidos de esa oración. Esto lo discutiré más ampliamente en la parte del modelo matemático.

Dadas dos oraciones $\mathcal{O}_1$ y $\mathcal{O}_2$, el resultado de aplicar α a $\mathcal{O}_1$ es otra oración $\mathcal{O}_2$ calculada de la siguiente manera:

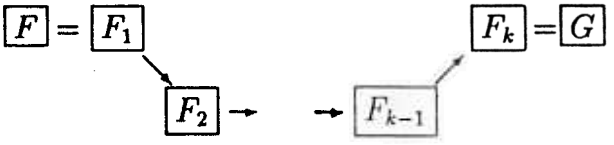
Inicialmente se unifica la frase $puedese(\alpha)$ con una subfrase de $\mathcal{O}_1$, si esto fuera posible, entonces la misma unificación se realiza en $añade(\alpha)$ y $borra(\alpha)$, Hecho esto, calculamos:

$$F_A = (\mathcal{O}_1 \oplus añade(\alpha)) \; y \; F_B = (\mathcal{O}_1 \ominus borra(\alpha))$$

para obtener que la oración O_2 es igual a $F_A \oplus F_B$.

Definición 3 *Una frase F* se sigue *de una frase G si hay una acción α, tal que podamos tener $\alpha(G) = F$.*

Definición 4 *Una frase G* se deduce *de una frase F si existe una sucesión $F_1, \ldots, F_k$ de frases, tal que $F = F_1$, $G = F_k$ y $\forall i$ tenemos que F_{i+1} se sigue de F_i.*



Vamos a decir que un átomo **r** es *activo* si existe una acción α tal que **r** se encuentra como parte de las precondiciones de la acción o como parte de sus postcondiciones. Y diremos que es *pasivo* en caso contrario.

Para cualquier frase F sea $F_a = \bigwedge\{x \mid x \text{ es activo y una instancia de él aparece en } F\}$. De igual manera $F_p = \bigwedge\{x \mid x \text{ es pasivo y una instancia de él aparece en } F\}$.

Así, podemos dividir una frase F en dos subfrases F_a y F_p (átomos activos y átomos pasivos).

Sean F y G dos frases tales que $F = (F_a, F_p)$ y $G = (G_a, G_p)$. Si $\exists x \in G_p$ tal que $x \notin F_p$, entonces G no se puede deducir a partir de F.

Vamos a convenir que *SOLAMENTE* se usarán símbolos de relaciones ACTIVOS en el problema de planeación.

Por ejemplo, si la descripción inicial del dominio contuviera sólo símbolos pasivos, no se requeriría de acciones para lograr cualquier otra descripción de ese mundo.

Esta convención nos lleva a tener definidas acciones que permiten lograr esos símbolos activos.

Condiciones de integridad.

Para el sistema lógico que estamos construyendo, necesitamos que nuestro lenguaje considere ciertas condiciones de integridad que le permitan reconocer inconsistencias propias del sistema y así evitar contradicciones.

Vamos a considerar dos tipos de condiciones de integridad. La primera es aquella que nos indica cuáles oraciones son *SIEMPRE verdaderas.* A estas condiciones se les llama "*Premisas Universales*" y se representan simbólicamente como:

```
always (oracion)
```

Las condiciones de integridad del otro tipo nos sirven para indicar de manera general cuáles oraciones no están permitidas en nuestro sistema. La generalidad se hace introduciendo variables, por lo que un grupo de oraciones queda representado por una frase. A estas condiciones les denominaremos "*Restricciones*" y las representaremos de la siguiente manera:

```
imposs (frase)
```

Así pues, la manera general de declarar a las oraciones que no pueden darse, es mediante frases, las cuales al instanciarse, nos dan las oraciones no permitidas.

2.1.3 El Modelo

El modelo solamente va a representar la parte del mundo real con la que se va a trabajar. A esta parte la llamaremos *El Dominio* del problema.

Todos los elementos que se encuentran y forman parte de este dominio son llamados *OBJETOS* y para describirlos vamos a usar los símbolos de objetos del lenguaje.

Por ejemplo, para el problema de los tres bloques, los objetos del dominio son únicamente los bloques y el piso. Para el problema de STRIPS, los objetos son: el robot, los cuartos, las cajas, el interruptor de luz, las puertas y los puntos que marcan una posición. En el caso del problema de la generación de código para una computadora, los objetos son los registros generales de la máquina y el acumulador. Las llantas, los ejes, los hoyos, el chasis, etc. lo son en el problema de ensamble de partes mecánicas.

Dentro del dominio, nos van a interesar ciertas relaciones que guardan algunos objetos con respecto a otros. Por ejemplo, la ubicación relativa entre dos o más objetos, la situación de un objeto, etc. Para representar el hecho de que la caja A está colocada sobre la caja B, podemos usar la relación **sobre(A,B)**. Para representar que el robot **R** se encuentra junto a la caja u objeto **A**, se usaría una relación como la siguiente **nextto(R,A)**.

Hemos mencionado que el mundo en cada instante se encuentra en una situación diferente. Los cambios de estado en nuestro modelo son debidos al efecto de acciones o movimientos sobre los objetos. Los diferentes estados los podemos representar mediante un conjunto de átomos instanciados. Por ejemplo, la situación que aparece en la figura 1.1, la podemos ver como el siguiente *conjunto de hechos*:

El bloque A está sobre el bloque B,
el bloque B está sobre el piso,
el bloque C está sobre el piso,
el bloque A no tiene nada encima,
el bloque C no tiene nada encima

Usando átomos podemos describir lo anterior de la siguiente manera:

$$sobre(A, B),$$
$$sobre(B, piso),$$
$$sobre(C, piso),$$
$$limpio(A),$$
$$limpio(C).$$

Obsérvese que esta situación está descrita solamente por *hechos*, en el sentido de nuestro lenguaje.

Por otra parte, tenemos que las relaciones entre los objetos son suceptibles de cambiar. Y para un momento dado, la situación de los objetos y las relaciones que guardan entre ellos, determinan lo que es un *ESTADO* del dominio.

Intuitivamente, los *estados* corresponden, en este modelo, a las situaciones en las que se puede encontrar el mundo en cuestión. Es decir, los estados modelan las situaciones del mundo real. Por lo tanto la interpretación de un estado la podemos ver como un conjunto de hechos

$$Estado := \{hechos\}$$

En efecto, los estados van a ser los elementos básicos de nuestro modelo, es decir vamos a considerar a *un modelo de estados.*

Vamos a tener una función de representación "**Rep**" que nos permite asociar oraciones con estados.

$$Rep : \{Oraciones\} \rightarrow \{Estados\} \tag{2.3}$$

La cual está definida como:

$$Rep(oración) = \{x \quad x \text{ es un átomo en la oración } \}$$

La función *Rep* nos hace manejar conjuntos de hechos, es decir, estados, como listas de hechos (oraciones).

Rep es una transformación suprayectiva, tal que, $\forall \mathcal{O} = a_1 \ldots a_m \in$ *Oraciones* y $\forall \Sigma = \{\sigma_1, \ldots, \sigma_k\} \in$ *Estados* se tiene que $\mathcal{O}$ *representa* al conjunto Σ, es decir, $Rep(\mathcal{O}) = \Sigma$ si y solo si,

i $\quad \forall \sigma_i \in \Sigma \; \exists j, \; i \leq j \leq m : \sigma_i = a_j,$

ii $\quad \forall j = 1, \ldots, m \quad a_j \in \Sigma$

Lema 1 *Dos oraciones representan el mismo estado si y solo si son equivalentes, en el sentido definido anteriormente* (Def. 2).

En efecto, basta observar que la aplicación sucesiva de las operaciones de conmutación, idempotencia y redundancia a oraciones no elimina átomos ni incrementa el número de átomos nuevos. Sólo afectan al número y orden de aparición de los átomos.

También podemos hablar de *SUBESTADOS* siendo éstos subconjuntos de un estado. Este concepto nos lleva a tener una relación reflexiva y transitiva, ($\leftarrow$) en el conjunto de las oraciones tal que, si $\mathcal{O}_1$ y $\mathcal{O}_2$ son dos oraciones cualesquiera, entonces :

$$\mathcal{O}_1 \leftarrow \mathcal{O}_2 \Leftrightarrow Rep(\mathcal{O}_1) \subset Rep(\mathcal{O}_2)$$

Se ve fácilmente que $\mathcal{O}_1 \leftarrow \mathcal{O}_2$, si y solo si una frase canónica equivalente a $\mathcal{O}_1$ es una sublista de una frase canónica equivalente a $\mathcal{O}_2$.

Desde un punto de vista puramente sintáctico, tenemos que nuestro conjunto de oraciones no es finito y tal vez podríamos pensar que lo mismo sucede con el conjunto de estados, lo cual nos produciría un modelo no finito. Sin embargo esto no es así, veamos porqué.

Primero, consideremos el conjunto $\mathcal{F}$ de todas las posibles frases formadas con todos los átomos instanciados, definidos para un dominio determinado. Tendremos que la cardinalidad de $\mathcal{F}$ es $\aleph_0$. Porque aunque el tamaño de las conjunciones de átomos es finito no así su número, pues podríamos tener frases con un mismo átomo repetido tantas veces como se quisiera. Por ejemplo, en el mundo de los bloques, las siguientes frases son válidas desde una perspectiva puramente sintáctica:

$$\begin{aligned} & on(a,b), \\ on(a,b) \;\&\; & on(a,b), \\ on(a,b) \;\&\; on(a,b) \;\&\; & on(a,b), \\ & \ldots etc. \end{aligned} \tag{2.4}$$

Esto representaría problemas para trabajar con los conceptos que se discuten más adelante, sobre consistencia y completitud de los sistemas de planeación, pues para ello se requiere que los conjuntos de estados sean finitos.

Por otra parte, tenemos que los estados son conjuntos de átomos instanciados (hechos), y es por ello, por ser conjuntos, que no existen elementos repetidos, como en los ejemplos (2.4).

También podemos considerar que las conjunciones que aparecen en la expresión (2.4) pertenecen a una misma clase de equivalencia, la cual, como se puede ver, representa a un mismo estado (Lema 1), áquel en donde el bloque a está sobre el bloque *b*: *on(a,b)*.

Vamos a ver con un ejemplo cuántos estados podemos tener para un dominio dado.

Consideremos el problema de los bloques, con n bloques en el dominio, sabemos que hay n^2 posibles átomos del tipo *on(X,Y)* y se pueden formar 2^{n^2} posibles relaciones con estos átomos. Algunas de estas relaciones son inconsistentes, si consideramos el mundo real, tal es el caso del átomo *on(x,x)*, que es un átomo inconsistente, porque en la realidad no podemos tener un bloque encima de sí mismo.

También, dado que las frases son conjunciones de átomos de longitud finita, tenemos que hay, para el problema de los bloques, n^{2k} frases de longitud igual a k. Estos números, aunque son o pueden llegar a ser muy grandes, son al fin y al cabo finitos.

Si vemos los elementos que conforman nuestro modelo, observamos que todos forman conjuntos finitos y que únicamente el conjunto de estados es el que ha quedado pendiente en saberse si es o no finito. Pero ahora, motivados por todo lo anterior, podemos establecer el siguiente:

Teorema 1 *El modelo así construído es finito siempre que el conjunto de símbolos de objetos y relaciones sea finito.*

Demostración:
Sea n la cardinalidad del conjunto de objetos y sea R el conjunto de símbolos de relaciones. Digamos que $\forall r \in R$ $ar(r)$ =*aridad de* r, entonces tenemos que hay $n^{ar(r)}$ posibles átomos completamente instanciados que tienen símbolo relacional r. En total tenemos que el número de átomos sin variables (instanciados) es de $A = \sum_{r \in R} n^{ar(r)}$. Este es el número de los átomos totalmente instanciados. Llamemos al conjunto de todos estos átomos instanciados como B. Ahora, si formamos un conjunto, S, con todos los posibles subconjuntos que se pueden formar con los elementos de B, tendremos que la cardinalidad de S es igual a 2^A. Esto nos indica que pueden haber 2^A posibles estados diferentes, desde el punto de vista puramente sintáctico. □

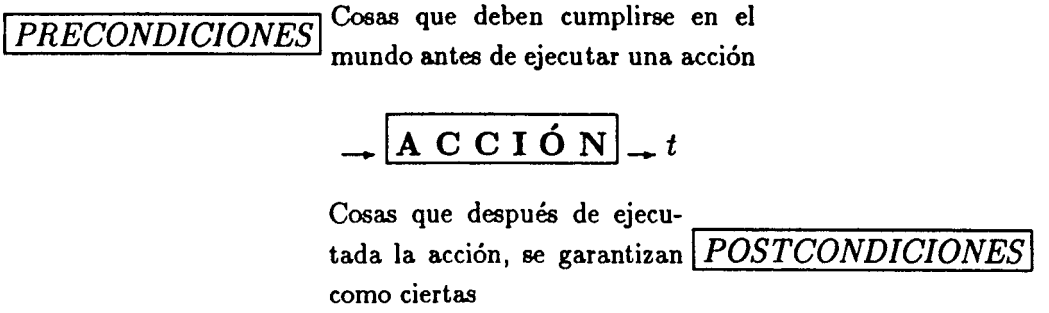
Dado que las relaciones entre los objetos pueden cambiar, tenemos que los estados cambian. Y en nuestro modelo vamos a considerar que los cambios de estado son debidos al efecto de algún agente que actúa directa y particularmente sobre los objetos, produciendo con ello el cambio de las relaciones. A estos agentes los llamaremos *ACCIONES.*

Para que una acción pueda aplicarse a un estado dado, deberán existir en ese estado ciertas condiciones previas antes de poderse efectuar el movimiento o acción. Por ejemplo, para

colocar un objeto adentro de una caja, ésta debe estar abierta antes de poderlo colocar. Pero también, un movimiento, después de haberse llevado a cabo, deja al mundo en una situación tal que es diferente, en principio, de la que tenía antes de llevarse a cabo el movimiento.

Las condiciones previas son un conjunto de hechos que cuando se presentan o se dan en un estado, entonces es posible aplicar la acción. A estas condiciones se le llama *precondiciones.*

Por otra parte, después de haberse llevado a cabo la acción, algunos hechos cambiaron y ya no existen más en el mundo, pero también otros fueron añadidos a la descripción del mundo. Estos hechos eliminados y añadidos por efecto de la acción, son llamados *postcondiciones.* Esto lo muestro esquemáticamente a continuación.



Tenemos entonces que cada acción tiene asociadas dos listas de hechos que corresponden a las precondiciones y las postcondiciones.

Para manejar esto, vamos a usar las funciones *puedese, añade* y *borra* de nuestro lenguaje, las cuales asocian a cada acción con sus precondiciones y sus postcondiciones.

Veamos un ejemplo sencillo de la aplicación de estos conceptos. Consideremos el ejemplo del "problema de los bloques", en el cual tiene definida una sola acción: **move(X,Y,Z)**. Recordemos que esta acción dice: "Mover el objeto X que se encuentra sobre Y a la posición encima de Z". Las precondiciones que deben existir para que esta acción pueda llevarse a cabo son:

1. El objeto X no debe tener nada encima.
2. El objeto X debe estar sobre Y.
3. Z no debe tener nada encima.
4. X y Z no pueden ser el mismo (cond. integridad)

Estos hechos los podemos representar mediante una conjunción de átomos de la siguiente manera:

$$clear(X),\ on(X,Y),\ clear(Z),\ X \neq Z$$

En la implementación en WARPLAN de estas ideas y utilizando el leguaje PROLOG, la función *puedese* se escribe como un predicado cuyo primer argumento es la acción y como segundo argumento está la lista de hechos que forman sus precondiciones.

Por lo tanto, tenemos que las precondiciones para la acción "move" pueden escribirse como:

$$can(move(X, Y, Z), clear(X) \wedge on(X, Y) \wedge clear(Z) \wedge X \neq Z)$$

Después de ejecutada la acción "move" se tiene garantizada la existencia de los siguientes hechos:

1. Y no tiene nada encima.
2. X está sobre Z.

Un hecho prevaleció: clear(X). Y otros hechos fueron eliminados: clear(Z), on(X,Y)
Entonces las postcondiciones de "move" son :

añade (clear(Y) $\wedge$ on(X,Z), move(X,Y,Z))
borra (clear(Z) $\wedge$ on(X,Y), move(X,Y,Z))

La acción "move" queda representada esquemáticamente de la siguiente manera:

PRECONDICIONES
clear(X), on(X,Y), clear(Z), X $\neq$ Z

ACCION
move(X,Y,Z)

POSTCONDICIONES
clear(Y), on(X,Z), $\neg$clear(Z), $\neg$on(X,Y)

En este ejemplo vemos que el hecho número 4 de las precondiciones es considerado, más que como una precondición, como una condición de integridad de la información que se está manejando.

Estas condiciones de integridad surgen del sentido común y permiten asegurar que las cosas que se están manejando bajo un esquema determinado no van a llevar a contradicciones.

En el contexto de la planeación tenemos dos condiciones de este tipo. Una es la que sirve para indicarle al sistema cuáles hechos *se cumplen para cualquier estado* en el que se

encuentre el dominio. La otra condición de integridad sirve para indicar cuáles hechos *no pueden darse bajo ninguna circunstancia* en el dominio.

Esta condiciones ya las discutimos en la parte del Lenguaje, y son las premisas universales y las restricciones.

La interpretación matemática en nuestro modelo de una premisa universal P es que si Σ es un estado, entonces puede sustituirse a Σ por $(\Sigma \cup Rep(P))\ \forall P$.

Para la otra condición tenemos que su interpretación es que si existe una restricción R, tal que $R \subset \Sigma$, entonces el estado Σ puede suponerse como inalcanzable o inconsistente ("inconsistente-2", ver sección 2.2).

Veamos con "el problema de los bloques" cómo se aplican estos conceptos de condiciones de integridad: En este dominio algo que siempre se cumple es que para cualquier estado, podemos colocar sobre el piso a los bloques, es decir, el piso siempre está "libre" o "clear", por lo tanto tenemos que una premisa universal para este problema es:

$$always\ (\ clear(piso)\ )$$

Por otro lado, algunas de las restricciones para este dominio son las siguientes:

- No se puede tener a un bloque encima de sí mismo.
- No se puede decir que un bloque Y está encima de un bloque X y tener que el bloque X está limpio.
- No es posible tener a un mismo bloque colocado encima de dos bloques diferentes al mismo tiempo.

Estas restricciones las podemos escribir de la siguiente manera:

$$imposs\ (\ on(X,X)\ )$$
$$imposs\ (\ on(Y,X) \wedge clear(X)\ )$$
$$imposs\ (\ on(X,Y) \wedge on(X,Z) \wedge Y \neq Z\ )$$

Las condiciones de integridad están muy relacionadas con el concepto de **consistencia** de un sistema formal, pues áquellas son utilizadas para investigar la consistencia del sistema, lo cual es uno de los aspectos fundamentales para obtener los resultados que se discuten en este trabajo de investigación. El tema de la consistencia lo trato en una sección posterior de este capítulo.

Por otra parte, vamos a decir que un estado Σ es *incongruente* si existe una oración $\mathcal{O}$ tal que $Rep(\mathcal{O}) = \Sigma$ y $\mathcal{O}$ es una instancia de una restricción. Y con esto tenemos que:

Definición 5 *Un estado es* PERMISIBLE *si no contiene algún subestado incongruente.*

Dos últimas observaciones sobre los estados son las siguientes:

1. Como el conjunto de estados es finito, entonces todo estado permisible está contenido en un estado permisible maximal.

2. Para cualesquiera dos oraciones $\mathcal{O}_1$ y $\mathcal{O}_2$ se cumple que:

$$Rep(\mathcal{O}_1 \oplus \mathcal{O}_2) \quad Rep(\mathcal{O}_1) \cup Rep(\mathcal{O}_2)$$

$$Rep(\mathcal{O}_1 \ominus \mathcal{O}_2) \quad Rep(\mathcal{O}_1) \setminus Rep(\mathcal{O}_2)$$

Sistema de Transición Estados-Acciones.

De la idea inicial que tenemos de un plan, se puede uno cuestionar que: Si un plan es una secuencia de acciones que transforman al dominio de estado en estado hasta obtener un estado final o meta, ¿cómo se puede pasar de un estado a otro mediante una acción ?

Para responder esta pregunta se define una relación de *transición* δ, tal que dado un estado Σ_1 produce un estado Σ_2 bajo el efecto de una acción α. Esto lo representamos simbólicamente como:

$$(\Sigma_1, \alpha) \xrightarrow{\delta} \Sigma_2 \tag{2.5}$$

Esta relación define un mapeo multivaluado:

$$\{estados\} \times \{acciones\} \rightarrow \{estados\}$$

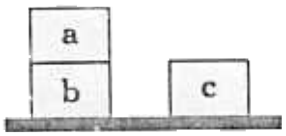
¿Cómo es que se puede dar esta transición? Veamos la siguiente:

Definición 6 *Dada* $\alpha \in$ {acciones} *y* $\Sigma \in$ {estados}. *Si existe una instancia a las variables en* α, *tal que el estado* T_0 *representado por la misma instancia de* puedese(α), *es un subestado de* Σ *(es decir, se cumplen las precondiciones de* α*), entonces se pasa o* transita *al estado* $(\Sigma \setminus T_2) \cup T_1$, *donde* T_1 =añade(α) *y* T_2 =borra(α) *con las mismas instancias. En este caso escribiremos:*

$$(\Sigma, \alpha) \xrightarrow{\delta} (\Sigma \setminus T_2) \cup T_1$$

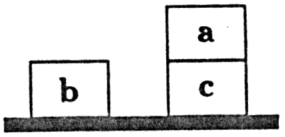
Observemos que de acuerdo a la anterior definición, la siguiente equivalencia es siempre cierta:

$$(\Sigma_1, \alpha) \xrightarrow{\delta} \Sigma_2 \Leftrightarrow \alpha(\mathcal{O}_1) = \mathcal{O}_2 \tag{2.6}$$



(a)

Figura 2.1: Estado inicial



(b)

Estado meta

En donde las oraciones $\mathcal{O}_1$ y $\mathcal{O}_2$ son las oraciones asociadas mediante la función **Rep** a Σ_1 y Σ_2 respectivamente: $Rep(\mathcal{O}_1) = \Sigma_1$ y $Rep(\mathcal{O}_2) = \Sigma_2$.

En la notación de WARPLAN, esto se escribe como: $(\alpha => U)$ (que es el estado resultante de aplicar α al estado U).

Voy a ejemplificar lo anterior, utilizando el problema de los bloques. Suponer que tenemos el estado inicial mostrado en la figura 2.1 a y que se ejecuta la acción de mover el bloque **a** encima del bloque **c** y llegar a tener el estado que se muestra en la figura 2.1 b.

Una representación del estado inicial Σ_1 sería la siguiente oración:

$$\mathcal{O}_1 = \text{on(a,b) \& on(b,piso) \& on(c,piso) \& clear(a) \& clear(c)}$$

Las precondiciones de la acción "move" (con sus variables ya instanciadas) son:

$$\gamma_0 = \text{clear(a) \& clear(c) \& on(a,b),}$$

(Nota: En este ejemplo no considero al átomo "$a \neq c$" por ser obvio, es decir, el bloque a es distinto del bloque c).

Es claro que, en efecto, $Rep(\gamma_0)$ es un subestado de Σ y por lo tanto es posible realizar la acción $\alpha = \text{move(a,b,c)}$.

Después de ser ejecutada la acción "move(a,b,c)" en el estado Σ, tenemos que la descripción del mundo estará dada ahora por el estado $\Sigma_2 = \Sigma_1 \setminus Rep(\gamma_2)) \cup Rep(\gamma_1)$, en donde

$$\gamma_1 = \text{añade}(\alpha) = \text{on(a,c) \& clear(b)} \quad \text{y}$$
$$\gamma_2 = \text{borra}(\alpha) = \text{on(a,b) \& clear(c)}$$

que son las postcondiciones de α. Y por tanto el estado Σ_2 es

$$\Sigma_2 = Rep(\text{on(a,c) \& on(b,piso) \& on(c,piso) \& clear(b) \& clear(a)})$$

Por otro lado, es claro que si tenemos un estado inicial y uno meta, no siempre es suficiente una sola acción para pasar de uno al otro. Vamos a ver ahora cómo se puede definir formalmente en nuestro modelo esta "transición" general.

Un estado está representado por una oración F, al aplicarle una acción α, se obtiene otro estado representado por una oración F', el cual en principio es un estado diferente. Vamos entonces a expresar la siguiente

Definición 7 *Un estado* Γ' **es accesible** *desde un estado* Γ, *si existe una sucesión de estados* $\Sigma_1, \ldots, \Sigma_k$, *y una sucesion de acciones* $\{\alpha_i\}$ *tales que:*

i $\Sigma_1 = \Gamma$, $\Sigma_k = \Gamma'$ *y*

ii $\forall i$ *se tiene que* $(\Sigma_i, \alpha_i) \xrightarrow{\delta} \Sigma_{i+1}$

Podemos ahora establecer el siguiente lema:

Lema 2 *Una oración* F_2 *se deduce de una oración* F_1 *si y sólo si el estado representado por* F_2 *es accesible desde el estado representado por* F_1.

Demostración:

Para la primera implicación tenemos que F_1 deduce F_2, entonces hay que demostrar que el estado representado por F_2, es accesible desde el estado representado por F_1.

Si F_1 deduce F_2 entonces existe la sucesión $(f_1, \ldots, f_k)$, en donde $F_1 = f_1$ y $F_2 = f_k$ y además $\forall i$ f_{i+1} se sigue de f_i, esto es, hay un conjunto de acciones $\{\alpha_i\}$ tales que:

$$\alpha_i(f_i) = f_{i+1} \ \forall i = 1, \ldots, k-1 \tag{2.7}$$

Como cada una de las f_i representa a un estado σ_i, entonces tenemos una sucesión de estados $\{\sigma_i\}$ tal que el estado σ_k representado por la oración f_k es accesible desde el estado representado por la oración f_1. Por lo tanto, el estado representado por F_2 es accesible desde el estado representado por F_1.

La demostración en el otro sentido es análoga a la argumentación anterior, sólo que en sentido inverso. □

Una reformulación del lema anterior la podemos establecer mediante el siguiente:

Corolario 1 *Sea* F *una oración inicial y* Σ *el estado que representa. Entonces para una oración cualquiera* G *se encontrará una secuencia de acciones* $\{\alpha_i\}$ *a partir de* F, *si y solo si, el estado* Σ', *representado por* G, *es accesible a partir de* Σ.

Podemos entonces ver a nuestro modelo como un sistema de transición no-determinístico.

2.1.4 Interpretación

En esta sección quiero enfatizar la interpretación que tienen los elementos del lenguaje formal sobre el modelo.

a Los *hechos* definidos en el lenguaje son instancias en el modelo, las cuales son hechos básicos del mundo real.

$$Hechos \longrightarrow Instancias$$

b Las *Oraciones* van a representar a los estados. Y se ve ahora globalmente que a cada situación del mundo real la podemos representar en nuestro modelo por medio de un estado y a éste lo interpretamos con una oración :

$$situación \mapsto estado \mapsto oración$$

Del Lema 1 tenemos que a oraciones equivalentes les corresponde la misma interpretación.

Transformaciones

De todo lo visto hasta ahora sobre las oraciones y los estados, podemos ver que dada una acción α, ésta define dos transformaciones. Una sobre las oraciones y otra sobre los estados. La primera es una transformación meramente sintáctica que actúa sobre las oraciones y la otra produce cambios de estados.

Consideremos que $T_{1,\alpha}$ es la transformación entre las oraciones:

$$T_{1,\alpha} : \{oraciones\} \rightarrow \{oraciones\}$$

Por ejemplo, sea la conjunción $g = on(a,b)\&clear(c)\&clear(a)$, la cual tiene asociada la oración $\gamma = (on(a,b), clear(c), clear(a))$ y sea $\alpha = move(a,b,c)$, entonces

$$T_{1,\alpha}(\gamma) = (clear(a), clear(b), on(a,c))$$

La segunda transformación vamos a representarla por $T_{2,\alpha}$. Esta es una funcion multi-valuada:

$$T_{2,\alpha} : \{estados\} \rightarrow \{estados\}$$

$T_{2,\alpha}(\Sigma)$ es un estado que se obtiene después de aplicar α al estado Σ, esto es,
$(\Sigma, \alpha) \xrightarrow{\delta} T_{2,\alpha}(\Sigma)$.

Por tanto, en un sistema de planeación tenemos que, para cualquier acción α existe una transformación $T_{1,\alpha}$ sobre oraciones y una transformación $T_{2,\alpha}$ sobre los estados, las cuales, junto con la *función de Representación* **Rep** (2.3), nos permiten tener el siguiente esquema de transformaciones:

$$\begin{array}{ccc} \{oraciones\} & \xrightarrow{T_{1,\alpha}} & \{oraciones\} \\ Rep \downarrow & & \downarrow Rep \\ \{oraciones\} & \xrightarrow[T_{2,\alpha}]{} & \{oraciones\} \end{array}$$

Y por ello y por las equivalencias probadas en esta sección, se cumple lo siguiente para cualquier acción α :

$$T_{2,\alpha} \circ Rep = Rep \circ T_{1,\alpha} \tag{2.8}$$

De todo este desarrollo teórico sobre los aspectos que involucran a los estados que modelan a las situaciones del mundo real podemos comentar que:

Al buscar y tener la conmutatividad expresada en (2.8) se garantiza que la manipulación simbólica que hace el sistema de planeación, tiene su equivalente en la descripción del mundo a través de los estados. Esto corresponde al problema expuesto al final de la motivación presentada en esta sección. Así, el modelo se hace mediante los estados, y sus transformaciones son manejadas mediante listas de átomos. Los resultados formales que obtenga una computadora con el lenguaje formal, efectivamente corresponden con lo que se quiere en el modelo.

2.1.5 Definición formal de la planeación

Hemos llegado finalmente a una definición formal de lo que es un plan:

Definición 8 *Dado un estado incial Σ_0 y un estado meta Σ_f. Un PLAN es una sucesión de acciones $(\alpha_1, \ldots, \alpha_k)$ tal que si $(\Sigma_0, \alpha_1) \xrightarrow{\delta} \Sigma_1$ y además $(\Sigma_i, \alpha_{i+1}) \xrightarrow{\delta} \Sigma_{i+1} \forall i = 1, \ldots, k-1$, entonces $\Sigma_k = \Sigma_f$. Gráficamente:*

$$\Sigma_0 \xrightarrow[\alpha_1]{} \Sigma_1 \xrightarrow[\alpha_2]{} \cdots \Sigma_{k-1} \xrightarrow[\alpha_k]{} \Sigma = \Sigma_f$$

Lo que importa realmente en un problema de planeación, es la sucesión de acciones α_i más que los propios estados intermedios por los que atraviesa el mundo.

De acuerdo al lema 2 y al corolario ?? tenemos las siguientes observaciones:

Un plan es una sucesión de acciones que hace accesible a Σ_i a partir de Σ_0

Un plan es una sucesión de acciones que transforman a cualquier frase que represente a Σ_0 en una frase que represente a Σ_f.

Algunos ejemplos de cómo se producen planes ya se han mostrado en el primer capítulo. En general la localizacion de planes se hace de manera puramente sintáctica. Asociado a cada Sistema de Planeación está el mecanismo que localiza planes. El que utilizo aquí lo presento en el capitulo 3.

2.2 Consistencia y Completitud

Los conceptos de consistencia y completitud se utilizan como propiedades de los sistemas formales dentro de las matemáticas [Hoftstadter80].

En el caso de los sistemas de planeación vamos a considerar que estos conceptos tienen, equivalentemente, los siguientes significados:

Consistencia: El sistema no demuestra proposiciones falsas, sólo proposiciones verdaderas (teoremas).
Completitud: El sistema no deja proposiciones verdaderas (teoremas) sin demostrar; se pueden construir las demostraciones de todas las proposiciones verdaderas.

Consistencia en los Sistemas de Planeación

La consistencia en los sistemas de planeación se considera sobre los estados más que en las acciones.

Vamos a decir que *un estado es inconsistente* si:

- NO es permisible.
- Es permisible pero NO es accesible desde un estado inicial

Dado que los estados se representan mediante frases, surgen dos tipos de problemas con respecto a la consistencia en los estados. Uno es decidir si una frase representa a un estado no permisible. Y el otro es decidir cuándo representa a un estado inaccesible.

El primer tipo se refiere a ver si un estado meta viola las premisas universales y/o implica las restricciones de integridad.

El otro tipo, el decidir cuándo un estado es accesible, es precisamente un problema de completitud.

Hay tres aspectos importantes que he considerado con respecto a la consistencia en los sistemas de planeación. Este asunto está muy relacionado con la decidibilidad [Hermes69] del problema de planeación.

El primer aspecto es que dada una meta, hay que saber a priori si ésta es consistente o no, de tal manera que si lo es, entonces se puede iniciar el proceso de planeación.

Una primera aproximación de esto, la hace WARPLAN. Simplemente vé si alguno de los átomos de la frase que representa la meta, es una instancia de una restricción de integridad. En caso afirmativo, se declara a la meta como imposible de lograr y termina el proceso de encontar un plan. De otra manera se iniciaría éste.

Visto esto como un procedimiento escrito en un lenguaje estructurado, tendríamos lo siguiente:

Si $C = (a_1, a_2, \ldots, a_k)$ es una frase, entonces el procedimiento para encontrar una inconsistencia debida a un átomo de C, sería algo como lo siguiente:

```
Begin
        consisten1 := true;
        for i := 1 to k do
                consisten1 := consisten1 and not (impossible(a_i)) ;
        if consisten1 then consistencia_primera
                      else inconsistencia_primera
end
```

En donde **consistencia_primera** indica que *todos* los átomos de la frase C son consistentes individualmente. El proceso "inconsistencia_primera" sirve para indicar que al menos uno de los átomos de C es inconsistente y por lo tanto el proceso de planeación no se inicia.

Este proceso tiene una complejidad que es lineal con k, esto es, $O(k)$. Y es sencillo de implementarse computacionalmente.

Una segunda aproximación, mejor que la anterior, revisa si algún subconjunto de átomos de C se hace una instancia de alguna restricción de integridad. El proceso de apareamiento sería como sigue:

```
Begin
        consisten2 := true;
        for each A ∈ Subfrases de C do
                consisten2 := consisten2 and not (impossible(⋀_{i∈A} a_i))
        if consisten2 then consistencia_segunda
                      else inconsistencia_segunda
end
```

El procedimiento **consistencia_segunda** nos indica que ninguna subfrase de C aparece como restricción en las condiciones de integridad.

En este caso el proceso "inconsistencia_segunda" sirve para indicar que al menos hay una subfrase de C tal que es inconsistente.

Este segundo proceso de revisión de consistencias tiene una complejidad exponencial con k, esto es, $O(2^k)$. Aunque es factible implementar este procedimiento de revisión de consistencias dentro de un planeador, es claro que repercute negativamente en el proceso de cómputo de un plan, resultando muy ineficiente el funcionamiento del planeador. Es

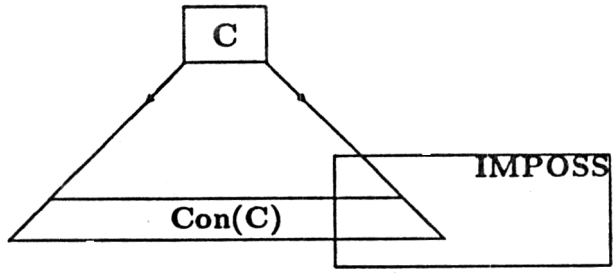


Figura 2.2: Esquema de los conjuntos **Con(C) e Imposs**

por ello que ningún planeador, considera la revisión de consistencias hasta este nivel. A cambio, lo substituyen con un mejor diseño de la base de conocimiento, poniendo atención principalmente en la descripción de las restricciones, de los predicados "imposs". Esto último no garantiza que el planeador no vaya a encontrarse con una inconsistencia generada durante el proceso de búsqueda de una solución, la que muy posiblemente no sea detectada como tal y haga caer al sistema en algún ciclo infinito.

Es claro que si tenemos una inconsistencia_primera entonces hay una inconsistencia_segunda, pues un sólo átomo se puede considerar como una subfrase, entonces:

$$inconsistencia\_primera \Rightarrow inconsistencia\_segunda \tag{2.9}$$

Los algorítmos anteriores son pruebas de consistencia de frases, para ver cuándo un estado no es permisible. Son algo burdos porque la selección de frases canónicas es un tanto arbitraria.

El tercer aspecto tenemos que considerarlo desde el punto de vista teórico y es cuando C forma un frase consistente de acuerdo a los dos casos anteriores. En este caso se considera al *conjunto de todas las consecuencias de C* , esto es, el conjunto de oraciones que se pueden generar, a partir de C, siguiendo las reglas de producción, es decir, las acciones, del sistema de planeación:

$$Con(C) = \{\mathcal{O} \in oraciones \quad \mathcal{O} \text{ se deduce de } C\}$$

Si el conjunto IMPOSS está formado por todas las conjunciones posibles que hacen instancias con el predicado "imposs", esto es, todas las frases que forman las restricciones de integridad, y si tuviéramos que la intersección IMPOSS $\cap$ $Con(C)$ es no-vacía, entonces la meta descrita por la frase C no se podría lograr, pues a partir de C podriamos a su vez lograr frases que fueran inconsistentes en el segundo sentido, ver figura 2.2. En este caso decimos que la frase C posee una *inconsistencia tercera.* Si la intersección anterior es vacía entonces decimos que la conjunción C tiene *consistencia tercera.*

De esto último podemos ver directamente que si tenemos una frase F con inconsistencia segunda (a priori), es decir, $F \in IMPOSS$. Y por otro lado F por sí misma está en $Con(F)$,

entonces tendríamos que una inconsistencia segunda implica que hay inconsistencia tercera. De aquí y con la expresión (2.9) tenemos que:

$$inconsistencia\_primera \Rightarrow inconsistencia\_segunda \Rightarrow$$
$$\Rightarrow inconsistencia\_tercera$$

Este último caso de inconsistencia no se puede llevar a una implementación práctica pues para ello sería necesario generar todos los posibles estados meta y resolverlos para obtener todos los C, sin considerar que pudieran encontrarse durante la búsqueda casos en donde el planeador cayera en ciclos. Es por ello que este problema es no-decidible a priori. Este tipo de consistencia puede ser evaluado en un sistema formal como el TWEAK [Chapman87]. El cual no es un sistema de planeacion en nuestro sentido dado su tamaño infinito. En el mismo trabajo se muestra que la detección de ciclos en un problema de planeación es indecidible. Así pues este tercer tipo de inconsistencia prácticamente no es decidible.

Podemos resumir estos tres tipos de inconsistencias diciendo que cualquier estado accesible es en efecto permisible, es decir, no viola las restricciones de integridad.

Completitud de los Sistemas de Planeación

La completitud en un sistema de planeación estaría definida de la siguiente manera:

Definición 9 *Un sistema planeación es completo si para toda meta que no sea inconsistente, puede encontrar un plan para lograr la meta. Es decir, en sistemas de planeación consistentes, para cualquier estado accesible se puede efectivamente encontrar un plan. En símbolos:*

$$not(inconsistente(meta)) \Rightarrow existe\_plan(meta)$$

Puede suceder que una meta expresada correctamente en la notación del sistema sea inconsistente, pero que el sistema no pueda detectarlo debido, principalmente, a que su conocimiento sobre las inconsistencias sea limitado o que también se involucren objetos que no existen para un problema particular. Y de esta forma el sistema "pensaría" que el planteamiento del problema es correcto (consistente) y trataría de resolverlo.

Muchos casos en donde los planeadores no pueden encontrar soluciones, se resuelven teniendo una buena y completa definición de las inconsistencias que hay dentro de un dominio dado. Pero, como hemos mencionado, no todos ellos se resuelven, y es precisamente aquí en donde la idea de reordenar los átomos de las frases que representan a un estado final dado, viene a resolver en gran medida algunos de esos casos.

Para un sistema de planeación dado y un problema dado podemos tener dos tipos de insolubilidad: En uno, la meta correspon de a un estado inaccesible, en otro la meta corresponde a un estado accesible pero el mecanismo de localización de planes del sistema de planeación, lo hace caer en un ciclo antes de encontrar efectivamente un plan.

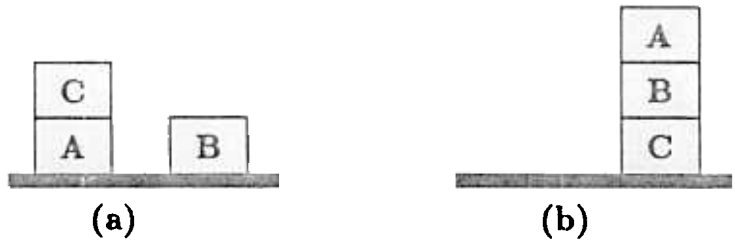


Figura 2.3: Estado inicial Estado meta

Una insolublidad del primer tipo se presenta con el "problema imposible" en el acertijo de los 15. Pues en este juego, dada una configuración inicial de los cuadritos, no se puede lograr la configuración en donde se encuentran colocados en orden descendente, del número 15 al 1.

Un sistema será tanto mas completo cuantos menos problemas sean insolubles en el segundo sentido.

El problema de completitud en los sistemas de planeación es sumamente importante. Vemos históricamente que a medida que se han ido resolviendo los problemas que los planeadores "veían" como insolubles (p.ej. anomalía de Sussmann), se han hecho más completos. Hasta llegar a TWEAK que es un sistema completo, pero en el cual se han tenido que sacrificar el determinísmo y la posibilidad de aplicarlo a situaciones prácticas del mundo real.

Con WARPLAN es posible establecer un balance entre un mecanismo determinístico y cierto grado de completitud, mediante la aplicación de heurísticas, pudiendo tener un sistema de planeación aplicable a diversos dominios prácticos.

En este trabajo presento un algoritmo que permite encontrar planes para metas con las que el planeador se ciclaba. Debido principalmente a que volvía a replantearse como meta intermedia la meta original (Secc. 2.3).

Tenemos que desarrollar mecanismos que vayan dirigiendo las búsquedas de trayectorias de estados y que hagan al sistema lo más completo posible.

2.3 Heurísticas para la consistencia y la reordenación

Un aspecto importante del trabajo realizado, fue el de encontrar que, alterando el orden en el cual se escribían los átomos de la oración que representaba a una meta dada, era posible obtener una solución a un problema de planeación que entraba en ciclos. Por ejemplo, consideremos el problema de bloques y supongamos que tenemos la situación que muestra la figura 2.3 a y queremos la situación que muestra la figura 2.3 b.

Si planteamos el problema de la siguiente manera: $on(a,c) \wedge on(c,b)$, el planeador entra en un ciclo tal que agota la memoria de la computadora. Pero si alteramos el orden de la meta de la siguiente manera: $on(c,b) \wedge on(a,c)$, entonces el planeador encuentra la solución

al problema, la cual es:

```
inicial;
move(a,b,piso);
move(c,piso,b);
move(a,piso,c).
```

El punto importante es entonces, determinar cuál es el orden apropiado que deben tener las submetas (hechos) dentro de la descripción del estado final o meta, esto es, la secuencia adecuada de permutaciones sobre este estado, para que un problema con ciclado tenga una solución.

Este reordenamiento de metas se efectúa en base a criterios y propiedades heurísticas del dominio que permiten tener una relación de orden entre los átomos que se utilizan para describir un estado determinado.

Función de jerarquía Q(x)

Consideremos una *función de jerarquía* Q, cuyo dominio es el conjunto de átomos que se usan para describir los estados (hechos). El contradominio de esta función debe ser un conjunto que tenga definida una relación de orden entre sus elementos. Este conjunto puede muy bien ser el de los números reales, con la relación de orden conocida.

Así Q es una función del conjunto de átomos al conjunto de los números reales:

$$Q(x) : \{\acute{a}tomos\} \rightarrow \mathrm{R}$$

Esta función de jerarquía nos permite hacer comparaciones entre los distintos átomos definidos para un mundo dado. Con ello podemos decir qué átomo tiene mayor o menor jerarquía que otro.

Lema 3 *Para todo conjunto de átomos g_i que tenga definida una función de jerarquía Q, cualquier estado $\Sigma = \mathrm{Rep}(\bigwedge_{i=1}^{n} g_i)$ tiene un estado equivalente $\Sigma^* = \mathrm{Rep}(\bigwedge_{i=1}^{n} g_i^*)$, en donde las g_i^* cumplen que:*

$$Q(g_1^*) \geq Q(g_2^*) \geq \cdots \geq Q(g_n^*)$$

Demostración:

Dado $Q : \{g_i\} \rightarrow \mathrm{R}$, si se ordena al conjunto $Q(g_i)$ en forma descendente se tiene: $Q(g_1^*) \geq Q(g_2^*) \geq \cdots \geq Q(g_n^*)$, en donde las g_i^* son los mismos elementos del conjunto $\{g_i\}$ pero en otro orden. La conjunción $\bigwedge_{i=1}^{n} g_i^*$, se puede ver como una permutación finita de $\bigwedge_{i=1}^{n} g_i$. Y por el lema 1, se tiene que $\sigma^* = Rep(\bigwedge_{i=1}^{n} g_i^*)$ es un estado equivalente al estado $\sigma = Rep(\bigwedge_{i=1}^{n} g_i)$. □

Niveles jerárquicos de Q(x,

Para definir la función de jerarquía Q, es necesario considerar dos niveles jerárquicos: uno *global* y otro *particular.*

Definición 10 *El Nivel Jerárquico Global es un número real positivo, con su parte decimal igual a cero, que se asocia con cada átomo del modelo.*

Por ejemplo, si tuviéramos los átomos: **empuja(A,B,R)**, **sobre(A,B)**, **junta(X,Y)**, **sube** y **baja**; una asignación de los niveles globales sería la siguiente:

sobre(A,B) → 1.0	sube → 4.0
empuja(A,B,R) → 2.0	baja → 4.0
junta(X,Y) → 3.0	

Es claro que podemos tener también dos o más átomos con el mismo valor jerárquico global. En el ejemplo anterior los átomos "sube" y "baja" tienen el mismo nivel.

El *nivel particular* es también un número real positivo que podemos considerar como un subnivel del nivel global de un átomo.

Pensemos que tenemos los átomos **sobre(x,y)** y **sobre(y,z)** . ¿Cuál es la relación de orden jerárquico que guardan entre ellos? Si tuvieramos una conjunción en la cual aparecieran estos átomos: *sobre(x,y)* ∧ *sobre(y,z)*, veríamos que em sobre(y,z) tiene más restricciones que *sobre(x,y)*, y por lo mismo es más conveniente lograrlo primero, antes que el otro. Es por ello que este átomo tiene mayor prioridad que el átomo *sobre(x,y)*. A este orden jerárquico es al que llamo *nivel particular* porque sólo se refiere al orden que tendrían los átomos con un mismo nombre de identificador (*Atomo General*). Por ello este nivel está relacionado con el nivel global del átomo en cuestión y se puede considerar como un subnivel del nivel global. Para esto, el nivel particular es un valor numérico decimal, cuya parte entera es precisamente el valor del nivel global del átomo general.

Siguiendo con el mismo ejemplo anterior, si al átomo general:*sobre(A,B)*, le corresponde un nivel global de 2.0, entonces, de manera particular los átomos del ejemplo: *sobre(x,y)* y *sobre(y,z)* ; podrían tener como valores jerárquicos 2.1 y 2.2 respectivamente, en donde los valores 0.1 y 0.2 son sus niveles particulares, los cuales son sumados al valor del nivel global para obtener su valor jerárquico final.

La definición formal de nivel particular es la siguiente:

Definición 11 *El Nivel Jerárquico Particular es un número en el intervalo abierto* $(0,1)$ *que se asocia a los átomos que tiene el mismo nombre, la misma aridad y el mismo tipo de argumentos, los cuales han sido ya instanciados con valores constantes. El valor jerárquico para estos átomos se forma sumando el valor del nivel particular al valor del nivel global del átomo general.*

Para definir la función Q sobre ambos niveles de jerarquía, es necesario hacer *consideraciones de caracter heurístico*, sobre el dominio que se esté trabajando. De hecho, la idea fundamental de este reordenamiento heurístico, se basa en la consideración de *tratar de resolver primero las metas más complejas e ir dejando al final las más sencillas.* En otras palabras, las consideraciones que deben hacerse son en el sentido de determinar cuáles son las condiciones que me permiten hacer primero una acción antes que otra, de tal manera que la obtención de una meta no enturbie o inhiba las metas posteriores.

Un algoritmo para definir la función Q se presenta a continuación:

ALGORITMO DE JERARQUIAS

Sea $A = \{a_1, \ldots, a_n\}$ el conjunto de todos los diferentes átomos de un dominio determinado. Y escribamos a cada átomo con sus respectivos argumentos así: $a_i = a_i(a_{i1}, \ldots, a_{im_i})$.

. Definir los valores numéricos de Q para cada uno de los elementos de A, y obtener de esta manera el conjunto $\{Q(a_i)\}_{i=1,\ldots,n}$, que son los valores jerárquicos del nivel global. Esto se hace mediante consideraciones heurísticas las cuales pueden ser:

- Determinar qué tipo de átomos deben lograrse primero antes que otros. De tal forma que al lograr una submeta descrita por un átomo no enturbie a las siguientes.
- Seleccionando los átomos que tiene mayor grado de dificultad para lograrse y asignándoles mayor jerarquía.
- Los átomos con más restricciones deben tener mayor valor jerárquico.

  (Recordar que estos valores jerárquicos tienen parte decimal igual a cero).

2. Si el átomo a_i tiene aridad diferente de cero, determinar sus valores jerárquicos del nivel particular. Esto se hace considerando dos instancias del átomo a_i :
   $a_i = a_i(a_{i1}, \ldots, a_{im_i})$ con $a_i = a_i(b_{i1}, \ldots, b_{im_i})$ y así poder determinar los niveles particulares, (también mediante consideraciones heurísticas) respondiendo a la pregunta: ¿ Cuál es la relación de orden que guardan $a_i = a_i(a_{i1}, a_{i2}, \ldots, a_{im_i})$ con $a_i = a_i(b_{i1}, b_{i2}, \ldots, b_{im_i})$?, ¿ Bajo qué circunstancias se da ? ¿ Cuándo y pórque se da ?

   El nivel particular es útil en el momento de plantear un problema de planeación concreto, pues sirve para ordenar los átomos de la conjunción (hechos) que representa una meta.

3. Si a_i tiene aridad cero, entonces su jerarquía queda definida únicamente por el valor del nivel global asignado al átomo en el paso (1), o sea, $Q(a_i)$.

4. Repetir 2 y 3 para cada uno de los átomos $\{a_i\}$.

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	

Figura 2.4: Acertijo de los 15

Antes de mostrar mediante algunos ejemplos, la aplicación de este algoritmo para ordenar una meta, quiero hacer unas consideraciones sobre la insolubilidad de los problemas de planeación y su relación con la jerarquización anterior.

En la sección anterior vimos que hay problemas de insolubilidad para los problemas de planeación. Uno de ellos nos indica que desde un punto de vista práctico hay metas que aunque sean consistentes no van a tener una solución dentro de un intervalo finito de tiempo, es decir, la cantidad de tiempo para encontrar una solución no está acotada superiormente.

Otro nos indica que desde un punto de vista teórico, hay problemas, que también son consistentes, pero que no tienen solución, tal es el caso de uno de los problemas típicos de planeación: el acertijo de los 15, en donde no existe ningún plan para lograr tener la configuración en donde los cuadritos se encuentran en orden descendente, a partir de ciertas configuraciones iniciales de los cuadritos. Esta configuración llamada "imposible" se muestra en la figura 2.4.

Y otro tipo de "insolubilidad" es debido al mecanismo de control cuando entra a ciclos infinitos sin lograr la meta. Este tipo de "insolubilidad" es posible salvarlo mediante la alteración del orden que tienen los átomos dentro de la conjunción que se utiliza para describir la meta.

La alteración del orden (reordenación) se hace precisamente considerando el orden descendente que se obtiene con la función de jerarquización aplicada a cada uno de los átomos de la meta. Este resultado se describe en el siguiente:

Enunciado de Reordenación: Si g es una frase que pasa la prueba de *consistencia segunda*, entonces, si fuera irresoluble del tipo 3, habría una reordenación de sus átomos dada por la jerarquización $Q(x)$, tal que replanteada así la meta, se encuentra un plan.

2.4 Localización de planes con la Jerarquización

Para mostrar la aplicación del algoritmo de jerarquización a casos concretos, voy a considerar algunos ejemplos típicos de planeación: uno es el problema de los bloques, (usando 5 bloques), cuyas bases ya se describieron en el capítulo 1 y otro ejemplo, también ya descrito, es el de STRIPS.

El problema de los bloques

Para este problema hay dos tipos de átomos definidos: *on(X,Y)* y *clear(X)* .

El primer paso es definir la función de jerarquía $Q(x)$ para este conjunto:

$$Q : \{on(X,Y), clear(X)\} \rightarrow \mathbf{R}$$

Desde un punto de vista heurístico se tiene que, el resolver un problema que involucre colocar un bloque encima de otro a partir de una configuración inicial dada es un problema más complejo que el problema de limpiar un bloque, esto es, quitarle de encima cualquier cantidad de bloques que tuviera. Pues simplemente se irían quitando uno a uno sin importar en donde quedan después (muy seguramente en el piso).

En este último caso, el planeador trataría, en primer lugar, de lograr la meta de limpiar el bloque en cuestión (bloque objetivo). Si éste no tuviera nada encima, sería un caso trivial, no haría nada. Si tuviera uno o más bloques encima de él, entonces, en primera instancia trataría de quitar el primer bloque que estuviera inmediatamente encima del bloque objetivo. Si éste a su vez tuviera otro bloque encima, este bloque se constituiría en la meta a lograr, es decir habría que quitarle de encima el bloque que tuviera. Y así sucesivamente hasta el último bloque de la columna. Finalmente se regresaría del proceso recursivo en el cual hubiera entrado y se resolvería así la meta original. Este proceso es en realidad más simple que el de construir una columna de bloques de acuerdo a un plan dado, mediante una descripción elaborada como una conjunción de átomos del tipo *on(X,Y)*.

Es por lo anterior que, heurísticamente, es más complejo, y por lo tanto tiene mayor jerarquía un átomo del tipo *on(X,Y)* que otro del tipo *clear(X)*. Por lo tanto tenemos que:

$$Q(on(X,Y)) > Q(clear(X)) \tag{2.10}$$

Asignamos números a estos átomos para tener definido el nivel global de la función $Q(x)$. Dado que solamente hay dos átomos en el dominio, una asignación natural sería que:

$$\text{on(X,Y)} \rightarrow 2.0$$
$$\text{clear(X)} \rightarrow 1.0$$

El paso 2 consiste en considerar dos grupos de argumentos para cada átomo y efectuar una comparación entre cada uno de ellos.

De acuerdo a esto, considérense los átomos: on(X,Y) y on(U,V), entonces la pregunta ahora es ¿ Cúal es la relación de orden entre las jerarquías de estos dos átomos ?

Para responder esta pregunta voy a introducir una definición de la función *sobre(X,Y)*, cuya idea intuitiva es que, X está sobre Y, si X se encuentra en la misma columna de bloques que Y, pero en una posición tal que está arriba de Y, aunque no necesariamente en la posición inmediata encima de Y. Su definición formal sería la siguiente:

Definición 12 *La función* sobre(X,Y) *es la cerradura transitiva del átomo* on(U,V), *esto es,* sobre = on^* .

Dos propiedades que involucran a la función *sobre* son:

$$\text{on(X,Y)} \Rightarrow \text{sobre(X,Y)}$$
$$\text{on(X,Z)}, \text{sobre(Z,Y)} \Rightarrow \text{sobre(X,Y)}$$

Regresando a la pregunta original, antes de la definición anterior, la respuesta es que:

$$Q(on(X,Y)) > Q(on(U,V)) \; si \; \vdash sobre(V,X) \tag{2.11}$$

esto significa que una meta, representada por el átomo *on(X,Y)*, tiene mayor jerarquía que otra representada por *on(U,V)*, si los bloques de esta última se encuentran por encima de los bloques X y Y, en una misma columna de bloques.

De esta manera queda determinada la jerarquía a nivel particular para los átomos del tipo *on(X,Y)*. Y esto debe ser considerado al momento de elaborar la estrategia de reordenación para los átomos dentro de un plan dado.

Para el átomo *clear(X)*, se procedería de la misma manera, es decir, se comparan dos átomos de la misma forma: *clear(X)* con *clear(Y)*, y se hace la misma pregunta: ¿ Cuál tiene mayor jerarquía ?

En realidad, el limpiar el bloque X o el bloque Y representa el mismo nivel de complejidad en ambos casos, dado que si se limpia primero X y luego Y o viceversa la complejidad es la misma y no interfiere una meta con la otra, pues se puede lograr muy bien una y luego la otra, considerando que para quitar un bloque de otro, áquel se coloca en primera instancia sobre el piso. Esto es indicado en la base de conocimiento.

Por lo tanto se concluye que la jerarquía a nivel particular para los átomos *clear(X)*, es única y es la misma para todos:

$$\text{Q(clear(X))} = \text{Q(clear(Y))}$$

Esto significa que no importa el orden en el que vayan los átomos de tipo *clear(X)* dentro de un plan, con respecto a sí mismos. Sin embargo sí deben de ir colocados después de los átomos de tipo on(X,Y), debido a la jerarquía definida anteriormente para el nivel global.

En este ejemplo no hay átomos con aridad igual a cero.

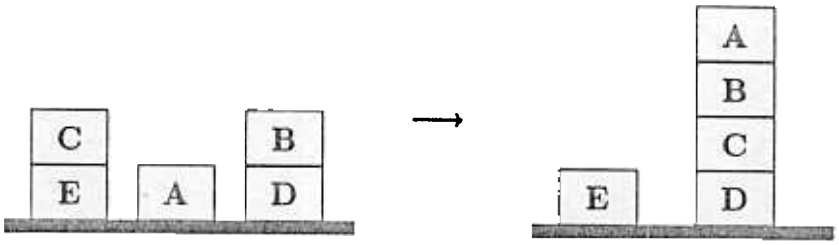


Figura 2.5: Un problema del mundo de los bloques

Vamos a aplicar lo anterior para resolver un caso simple. Consideremos que se tiene el estado inicial mostrado en la figura 2.4 y que se quiere obtener el plan para lograr el estado meta, que se muestra a la derecha de la misma figura.

El estado meta se describe mediante la siguiente conjunción de átomos:
clear (E)& on(A,B) & on (B,C) & on (C,D).

Ahora hay que determinar el orden en el que deben ser colocados los átomos dentro de la conjunción. Para ello, observamos que hay un sólo átomo del tipo *clear(X)*. De acuerdo a (2.10), este átomo debe ir después de los otros átomos. Como el otro átomo que aparece en la conjunción es *on(X,Y)*, hay que determinar la jerarquía del nivel particular.

Para ello usamos la desigualdad (2.11), que en el presente caso se cumpliría de la siguiente manera:

$$Q(\text{on(C,D)}) > Q(\text{on(B,C)}), \text{ porque} \vdash \text{sobre(B,D)}$$
$$Q(\text{on(B,C)}) > Q(\text{on(A,B)}), \text{ porque} \vdash \text{sobre(A,C)}$$

Por lo tanto reescribiendo la meta en orden descendente, de acuerdo a la jerarquización a nivel particular y al nivel global, se tiene:

$$\text{Meta: on(C,D) \& on(B,C) \& on(A,B) \& clear(E)}$$

En esta forma habría que proporcionarle la meta al planeador, para encontrar un plan que la obtenga.

WARPLAN produce el siguiente plan como salida:

```
<< PLAN >>

move(B,D,piso);
move(C,E,D);
move(B,piso,C);
move(A,piso,B).
```

El Mundo de STRIPS

El mundo de este problema [Fikes 71] consiste de un ambiente cerrado en el cual hay cinco cuartos. Cuatro de ellos están conectados mediante el cuarto 5, que sirve como corredor. Ver la figura 1.3.
Los átomos en este problema son los siguientes:

at(robot,P)
nextto(robot,X)
nextto(X,Y)
status(S,on)
on(robot,B)
onfloor
inroom(robot,R)

A continuación voy a hacer un análisis heurístico de las acciones y hechos involucrados en este mundo para poder determinar los valores jerárquicos de la función $Q(x)$.

Hay 6 átomos diferentes y uno de ellos: nextto, tiene 2 formas. Ahora aplicando el algoritmo de jerarquías para definir la función $Q(x)$ para este conjunto de átomos, se tiene:

Analizando cada una de las 7 situaciones (átomos), se puede ver que la orden de encender el interruptor, es la tarea con mayor grado de dificultad, pues para efectuarla, el robot debe colocar una caja en la posición 4 y luego subirse para poder encender el interruptor. Por tanto Q(status(S,on)) es el máximo valor de la función, que podemos asignarlo igual a 7 (que es el máximo número de átomos diferentes en el conjunto).

La siguiente tarea con mayor grado de dificultad es aquella en la que se deben colocar juntos los bloques. Por tanto Q(nextto(U,V)) es el siguiente valor abajo del máximo, esto es, igual a 6.

Aparentemente el subirse y el bajarse de un bloque, tienen la misma complejidad en este contexto. Pero no es así, pues para que el robot se suba a una caja primero tiene que trasladarse hasta donde está la caja y luego subirse. En cambio bajarse no requiere de otra cosa que estar arriba de ella. Por lo tanto:

$$Q(\text{ on(robot,B) }) > Q(\text{ onfloor })$$

Aunque su valor todavía no se puede determinar, sino hasta analizar los átomos restantes.

Por otra parte, hay 2 átomos que involucran acciones muy semejantes, ellos son inroom(robot,R) y at(robot,P). Ambos indican que el robot se traslade a un sitio determinado, que en el primer caso es un cuarto y en el otro es un punto cualquiera dentro del espacio del ambiente. Sin embargo, es posible ver que el segundo átomo es un poco más complejo de realizar que el primero, pues inclusive el primero es una parte del segundo, dado que éste puede indicar un punto en cualquiera dentro de los cuartos, no solamente el cuarto mismo como es la acción del primer átomo. Por esto, llegamos a que las jerarquías de estos átomos cumplen que:

$$Q(\text{ inroom(robot,R) }) < Q(\text{ at(robot,P) })$$

Comparar estas dos situaciones con las de subirse y/o bajarse de una caja, requiere de una consideración particular. Como ya se vió, bajarse es más sencillo que subirse. Sin embargo para subirse a una caja, el robot debe encontrase en el sitio en donde está la caja y por lo tanto debe trasladarse hasta ella. Esto implica que tal acción involucra al átomo "inroom", por lo que si se tuvieran estos dos átomos como parte de una conjunción de metas o átomos, sería conveniente satisfacer primero el átomo "inroom", pues ello facilitaría las cosas a la acción de subirse. Es claro que estas dos acciones deben ser consistentes (esto lo detecta el planeador mediante las reglas de consistencia que maneja), esto quiere decir que la caja debe estar dentro del cuarto al que se va a trasladar. Por lo tanto tenemos que:

$$Q(\text{ on(robot,B) }) < Q(\text{ inroom(robot, R) })$$

Y esto nos conduce (uniendo las tres últimas desigualdades) a:

$$Q(\text{ onfloor}) < Q(\text{ on(robot,B) }) < Q(\text{ inroom(robot,R) }) < Q(\text{ at(robot,P) }) .$$

Finalmente, dado que ya se habían determinado las jerarquías para "nextto" y "status" las cuales eran las mayores del conjunto de átomos, llegamos a que $Q(x)$ cumple con:

Q : status(S,on) $\rightarrow$ 7 ;
Q : nextto(U,V) $\rightarrow$ 6 ;
Q : at(robot,P) $\rightarrow$ 5 ;
Q : inroom(robot,R) $\rightarrow$ 4 ;
Q : on(robot,B) $\rightarrow$ 3 ;
Q : onfloor $\rightarrow$ 2.

De esta manera quedan determinadas las jerarquías a nivel global.

Ahora sigue el paso 2 del algoritmo, el cual consiste en hacer comparaciones para cada uno de los átomos.

El primer átomo, "status(S,on)", no tiene mayor problema dado que en este ambiente solamente hay un interruptor S, y entonces nó hay otra expresión que comparar.

El siguiente átomo es "nextto", el cual tiene dos formas:

$$\text{nextto(robot, X) y nextto(X, Y),}$$

Para el primero se considera otra expresión y se comparan:
nextto(robot,X) y nextto(robot,Y). En este ejemplo X y Y representan cajas. Como el robot es único, no puede estar en dos posiciones diferentes al mismo tiempo, por lo que no procede la comparación y esta situación es prevenida mediante el mecanismo de consistencia del sistema, colocando esta información en la base de conocimiento.

La segunda forma tampoco representa mayor problema, pues el reunir dos cajas no interfiere o tiene mayor importancia que el reunir otras dos cajas diferentes. Por lo tanto tampoco esta forma tiene una jerarquía a nivel particular. Sin embargo falta la comparación cruzada de las formas, esto es, hay que comparar las dos formas de "nextto" entre ellas mismas.

La forma "nextto(X,Y)" requiere que el robot actúe sobre una de de las cajas, empujándola hasta donde está la otra caja. La otra forma solo indica que el robot debe colocarse junto al objeto X. Resulta claro que la primera forma debe ejecutarse primero, porque es más compleja que la otra, o sea, debe tener precedencía sobre la última. Luego entonces se tiene que:

$$\text{nextto(robot, X)} < \text{nextto(X, Y)}$$

Tanto para el siguiente átomo, "at(robot,P)",como para "inroom(robot,X)", la argumentación para determinar la jerarquía a nivel particular, es la misma que aquella para la forma "nextto(robot,X)" dado que tiene casi la misma semántica. Por lo tanto estos átomos se quedan con su jerarquía a nivel global.

Para el átomo "on(robot,B)", también el argumento es parecido. Dado que el robot es único y sólo puede estar arriba de una caja a la vez, entonces no procede la comparación. Y si se llegara a presentarse la situación de subir al robot a dos cajas distintas al mismo tiempo, el mecanismo de consistencias lo detectaría y lo marcaría como un error. Entonces la jerarquía de este átomo es la del nivel global.

Finalmente, el último átomo "onfloor", no tiene argumentos y por el paso 5 del algoritmo, se queda con su valor de la jerarquía a nivel global. Por lo que el arreglo final de jerarquización de los 7 átomos de este problema sería:

Q : status(S,on) → 7 ;
Q : nextto(U,V) → 6.2 ;
Q : nextto(robot,U) → 6.1 ; Q : at(robot,P) → 5 ;
Q : inroom(robot,R) → 4 ;
Q : on(robot,B) → 3 ;
Q : onfloor → 2 .

Vamos a aplicar este conocimiento a un problema particular. Supongamos que se quiere que el robot coloque las tres cajas juntas, que encienda el interruptor y que se coloque en el punto 6. La descripción de la meta podría ser la siguiente:

```
Meta:     nextto(box(1),box(2)),   { Coloca las cajas juntas }
          nextto(box(2),box(3)),
          status(S(1),on),         { Enciende el interruptor }
          at(robot,point(6)).      { Transladarse al punto 6 }
```

Si esta meta se escribe en el mismo orden en el que aparece, el planeador entrará en un ciclo infinito, es decir hay un "caso singular" y no encontrará el plan. La razón de esto es que,

para lograr la meta planteada, procede como sigue: Las dos primeras submetas producen el siguiente plan parcial:

```
inicial;
goto2(box(2),room(1));
pushto(box(2),box(3),room(1));
goto2(box(1),room(1));
pushto(box(1),box(2),room(1);
...
```

Para resolver la siguiente parte de la meta, es decir, "status(S(1),on)", se hace mediante la acción "turnon(S(1))", cuyas precondiciones son: "on(robot,box(1)) & nextto(box(1),S(1))"

El primer átomo de las precondiciones se cumple sin ningún problema. Pero el segundo, tiene que mover la caja 1 hasta la posición del interruptor S(1). Esto lo hace con la acción "pushto(box(1),S(1),room(1))", que dice que "empuje la caja 1 hasta S(1), en el cuarto 1". Con esto se destruye una de las submetas ya logradas y no puede aplicar el método por extensión. Ni tampoco puede aplicar el método por inserción, debido al axioma de este método (capítulo 3; *"Funcionamiento de planeador"*), pues si vemos la última acción del plan parcial: "pushto(box(1),box(2),room(1)", elimina a la acción de la submeta que estamos tratando de lograr, "pushto(box(1),box(2),room(1)".

Sin embargo si ordenamos de acuerdo a las jerarquías anteriormente encontradas, la meta se escribiría con el siguiente orden:

```
Meta:    status(S(1),on),          [7]
         nextto(box(1),box(2)),    [6.2]
         nextto(box(2),box(3)),    [6.2]
         at(robot,point(6)).       [5]
```

Lo que aparece entre paréntesis cuadrados son los valores de jerarquía respectivos, y como puede verse los átomos correspondientes están colocados en orden decreciente, de acuerdo con su valor jerárquico.

La meta escrita de esta manera conduce a que el planeador encuentre una solución para este problema:

```
<< P L A N >>

inicial ;
goto2(box(2), room(1)) ;
pushto(box(2), box(3), room(1)) ;
goto2(box(1), room(1)) ;
pushto(box(1), lightswitch(1), room(1)) ;
```

```
climbon(box(1)) ;
turnon(lightswitch(1)) ;
climboff(box(1)) ;
pushto(box(1), box(2), room(1)) ;
goto2(door(1), room(1)) ;
gothru(door(1), room(1), room(5)) ;
goto2(door(4), room(5)) ;
gothru(door(4), room(5), room(4)) ;
goto1(point(6), room(4)).
```

Ensamble de partes mecánicas

El problema de emsamble de partes ya lo traté en el primer capítulo. En esta sección voy a ampliar la descripción del problema, viendo la definición de los átomos que representan a los hechos y de las acciones involucradas .

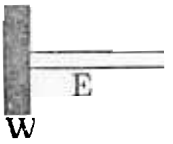
Hay definidos 11 átomos[1], con sus respectivas acciones para obtenerlos:

HECHO	ACCION
W is_attached_to E	insert E into W
A is_thru H	slide D1 end_of A into H from D2
W is_clamped	clamp wheel W
axle A is_clamped	insert end_of axle A into W
wheel W is_free	unclamp wheel W
axle A is_free	unclamp axle A
vice is_free	unclamp X
car_body_is_blocked_to D	block_car_body_to D
car_body_is_unblocked_to D	unblock_car_body_to D
D1 end_of A points D	slide D1 end_of A into H from D2 D is_opposite _of D2.
D end_of A points D2	slide D1 end_of A into H from D2 D is_opposite _of D1.

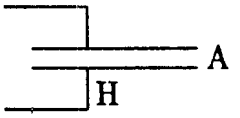
Gráficamente podemos considerar que los algunos de los átomos anteriores tienen las siguientes interpretaciones esquemáticas.

[1]La notación es inglés porque así se manejo el ejemplo con los programas de cómputo

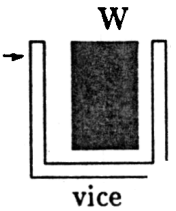
"W is_attached_to E"



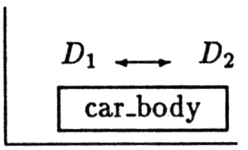
"A is_thru H"



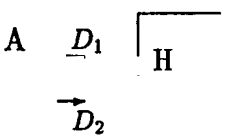
"W is_clamped"



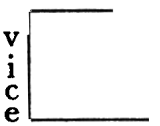
"car_body_is_blocked_to_D_1"
"car_body_is_unblocked_to_D_2"



"D_1 end_of A points D"
" D end_of A points D_2"



"axle A is_clamped"
"axle A is_free"



Primero procedemos a asignar los valores de Q para cada uno de estos átomos. Para ello debemos tener claro qué significa cada uno de estos hechos. ¿ Cuál es el más restrictivo de todos ?, ¿Qué situación debe tenerse antes que cualquier otra?, etc.

La lista de hechos incluye: Que una rueda esté colocada al extremo de un eje; que un eje se encuentre a través de un orificio del chasis; que una rueda o un eje estén sujetados por la pinza o el chasis, respectivamente, o que estén libres; que el chasis esté asegurado o liberado con respecto a una base. Y otros dos que indican en qué dirección apuntan los extremos de un eje.

De lo anterior podemos comenzar a asignar las jerarquías: Como son 11 los átomos, asignemos el número 11 al átomo de mayor jerarquía.

Este valor debe corresponder a los átomos que indican la dirección que deben tener los ejes, pues sería muy inconveniente que ya colocados los ejes, al final tuvieramos que orientarlos de acuerdo a lo que dice este átomo. Por lo tanto Q(D1 end_of A points D)=11, y también Q(D end_of A points D2) = 11.

Luego el siguiente hecho es tener colocado el eje dentro del chasis, antes de cualquier otra situación. Por ello tenemos que Q(A is_thru H)=10.

átomo también se queda con su nivel global.

Por lo tanto la jerarquización obtenida en el nivel global es la jerarquización final para los átomos de este mundo.

2.5 Eficiencia de los planes con la jerarquización

De los ejemplos anteriormente tratados, alguien podría decir que para cada uno de los mundos definidos y los problemas que se presentan y se resuelven ahí, hay un algoritmo que proporciona las soluciones a tales problemas. Y en efecto, muchos de los problemas de la planeación pueden resolverse mediante algoritmos particulares para cada uno de esos mundos.

De hecho, en términos generales se puede decir que cualquier problema que tenga solución, es susceptible de resolverse mediante la aplicación de algún algoritmo. Y los problemas típicos que se manejan en planeación, no son una excepción.

Sin embargo, hay que recordar que los actuales planeadores son el producto del desarrollo de los programas de cómputo denominados, dentro de la inteligencia artificial, "Resolvedores de Problemas". Uno de los primeros y más distinguidos de estos programas fue el *"General Problem Solver"* [Newell63].

Un planeador independiente del dominio, considera cualquier mundo sin importar de que se trate. Así, puede resolver problemas de robótica como de logística. Por su misma generalidad es probable que algunas veces pueda producir planes que tengan algunas acciones redundantes o sin sentido dentro del plan. Pero ésto se puede ir eliminando conforme se vayan produciendo planes y el usuario detecte estas situaciones, pudiéndolas cambiar luego, mediante la modificación adecuada de la base de conocimiento o utilizando técnicas como la jerarquización aquí propuesta.

La reordenación de las metas mediante la función de jerarquía $Q(x)$ está basada en conceptos humanos para resolver problemas. Por ejemplo, en el problema de los bloques, un posible algoritmo que resuelve cualquier problema que sea consistente en este mundo, es aquel que dice que se deben colocar todos los bloques sobre el piso y luego colocarlos para ir formando la pila (esta colocación de una u otra forma tiene implícito un orden basado en qué es primero y qué es después). Sin embargo una persona no resuelve así el problema, sino que va viendo qué bloques deben ir colocados en el orden requerido, resolviendo primero una meta y luego otra, que no necesariamente es la que sigue en la descripción dada del estado final. Lógicamente, primero resolverá los bloques que están más abajo de la pila e irá colocando los bloques que siguen hacia arriba.

Lo que se pretende con el algoritmo de jerarquización es "dar" el conocimiento que tiene una persona para resolver los problemas como ella lo haría, al planeador para que lo haga igual y de acuerdo a esos criterios.

Es fácil ver que si aplicamos el algoritmo anterior para resolver los problemas del mundo de los bloques, el número de movimientos requeridos es mayor que si lo hacemos como sabemos.

El planeador trabajando con la jerarquización produce además planes cuya longitud es menor que aquellos producidos siguiendo un algoritmo basado en pasar primero a un "estado base" para que a partir de él se logre la meta final, lo cual tendrá implícito un ordenamiento tal que las sub-metas logradas no intefieran a las subsecuentes, que es precisamente lo que se pretende con la jerarquización.

De todas las corridas hechas con el planeador y utilizando la función de jerarquización obtuve que el 99% de los planes producidos tenían longitud mínima ("planes óptimos"). El otro 1% no se obtuvo debido a cuestiones lógicas, las cuales explico detalladamente en el siguiente capítulo, en la sección de experimentos y en otros casos debido a una mala ubicación de las clausulas en la base de conocimiento (recordar que trabajé con PROLOG, y el orden de las clausulas es importante).

2.6 Completitud de los sistemas de planeación

En esta sección doy algunas ideas sobre la completitud de los sistemas de planeación bajo el enfoque de las heurísticas de reordenación de metas y consistencia de las mismas.

Consideremos el conjunto de todas las conjunciones de átomos o frases que se puedan formar para un sistema de planeación no-lineal conjuntivo, como los que he estado discutiendo en este trabajo. Sea este conjunto $\mathcal{F}$. En la presente discusión se considerará que las frases son cada una, elementos únicos tomados de la clase de equivalencia que representa a un estado único determinado, tal como se vió en el capítulo 2 (secc.2.4).

Este conjunto está dividido en dos subconjuntos principales: El de las conjunciones que son consistentes, de acuerdo a los criterios anteriormente descritos en la sección 2.2. Y el otro es el que contiene a todas aquellas frases que son inconsistentes. Denotemos a estos conjuntos por $\mathcal{C}$ e $\mathcal{I}$, respectivamente. Así tenemos que $\mathcal{F} = \mathcal{C} \cup \mathcal{I}$.

Para un sistema de planeación determinado, se tendrán definidos algunos predicados "imposs", que generan algunas frases del conjunto $\mathcal{I}$. Sea $\mathcal{B}$ el conjunto generado por "imposs".

El sistema también podrá demostrar algunas frases que sean consistentes y generará un conjunto de frases que son demostrables. Llamemos a este conjunto $\mathcal{A}$. Estos conjunto se muestran en la figura 2.6.

El subconjunto $\mathcal{A}'$ está formado por aquellas conjunciones de átomos que siendo consistentes de acuerdo a las reglas de consistencia del sistema, no se pueden demostrar o, en otras palabras, que bajo un esquema de consistencia dado no existe un plan que las haga verdaderas.

$\mathcal{F}$ es un conjunto muy grande, pero finito, que se puede enumerar y más o menos se pueden determinar $\mathcal{C}$ e $\mathcal{I}$ si los postulados son verdaderos y por lo tanto consistentes.

En la investigación que llevé a cabo encontré dos resultados importantes en relación a estos conjuntos:

Primero, que una definición *completa* del predicado "imposs", esto es, haciendo que $\mathcal{B}$ cubra totalmente a $\mathcal{I}$, hace que el subconjunto $\mathcal{A}$ tienda a cubrir totalmente al $\mathcal{A}'$. Esto se

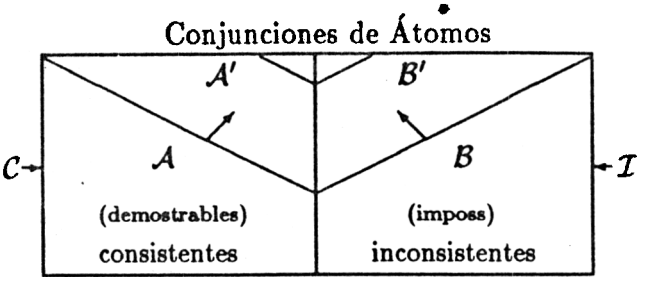


Figura 2.6: Conjunto de frases, $\mathcal{F}$

demuestra adelante.

Y segundo, que una reordenación jerárquica, basada en consideraciones heurísticas, es decir, alimentando al sistema con conocimiento humano para resolver problemas no determinísticos, permite hacer demostrables los elementos de $\mathcal{A}'$ (*"Enunciado de reordenación"*).

Para tratar el primer aspecto matemáticamente, se define una función ϕ que depende de la jerarquización P y de una manera indirecta, de la consistencia definida para un sistema dado. Esta función se define como el cociente entre el número de las conjunciones que son demostrables (cardinalidad de $\mathcal{A}$) y el número de las que son consistentes (cardinalidad de $\mathcal{C}$):

Definición 13 *El Cociente de Demostrabilidad es la relación entre la cardinalidad del conjunto de frases demostrables y la cardinalidad del conjunto de frases consistentes en un sistema formal. Y se expresa de la siguiente manera:*

$$0 \leq \phi(P) = \frac{\text{demostrables}}{\text{consistentes}} \leq 1$$

Es claro que este cociente está acotado por cero y uno, pues el número de conjunciones demostrables será a lo más, igual al número de conjunciones consistentes posibles (todo el conjunto $\mathcal{C}$). Si $\phi = 0$ quiere decir que nada se puede demostrar.

De la misma manera se puede introducir una medida para la inconsistencia. Definiendo una función cociente ψ entre el número de conjunciones (*n-imposs* = cardinalidad de $\mathcal{B}$) generadas por los "imposs" definidos en la base de conocimiento del sistema, y el número total de conjunciones que son inconsistentes (*n-inconsist* = cardinalidad de $\mathcal{I}$), que es el número de elementos del conjunto formado por todas las frases que pertenecen a cualquiera de las tres clases de inconsistencias, (ver sección 2.2):

Definición 14 *El Cociente de Imposibilidad es la relación entre la cardinalidad del conjunto de frases generadas por los predicados "imposs" de un sistema de planeación, y la cardinalidad del conjunto de frases que son inconsistentes para tal sistema:*

$$\psi = \frac{\text{n} - \text{imposs}}{\text{n} - \text{inconsist}}$$

Al construir un sistema de planeación, casi siempre se tiene que $\mathcal{B}' \neq \{\emptyset\}$, en las primeras etapas de su funcionamiento y conforme va madurando el sistema con experiencias y corridas, $\mathcal{B}$ comienza a crecer hasta cubrir toda la parte derecha de $\mathcal{F}$, haciendo que $\mathcal{B}' \simeq \{\emptyset\}$.

Mientras que $\mathcal{B}$ no cubra a todas las conjunciones inconsistentes, es posible que al sistema se le proporcionen frases del subconjunto $\mathcal{B}'$ provocando con ello que no encuentre soluciones para metas que contengan tales frases.

Mediante ampliaciones a la definición de "imposs" es posible cubrir todo el subconjunto de inconsistencias $\mathcal{I}$. Por esto el primer resultado me conduce a establecer el siguiente teorema:

Teorema 2 *Para un sistema de planeación no-lineal conjuntivo, el cociente de demostrabilidad es aproximadamente igual a 1, si y solo si el cociente de imposibilidad es también cercano a 1.*

Demostración:

Supongamos que tanto ϕ como ψ tienen unos valores de ϕ_1 y ψ_1 respectivamente. $\psi_1 < 1$, esto significa que $\mathcal{B}' \neq \{\emptyset\}$. Sea f una frase de $\mathcal{B}'$ y considérese una meta g que contenga a f en su descripción. Entonces el sistema al tratar de demostrar g, va a fallar dado que existe una inconsistencia en la descripción del problema.

Ahora vamos a marcar a f como una frase inconsistente mediante un predicado "imposs" que produzca a f, y tal vez a más frases semejantes. Esto hace que $\mathcal{B}$ tenga más elementos, obtenidos de $\mathcal{B}'$, teniendo ahora ψ un nuevo valor ψ_2 que es mayor que ψ_1. Con ello se consigue que $\mathcal{B}$ sea más parecido a $\mathcal{I}$, pero no igual, todavía.

Al estar f en $\mathcal{B}$, ya no puede ser considerada como parte de una meta a demostrar, por lo que ahora se pueden considerar conjunciones de átomos que no contengan a frases del tipo f, las cuales se pueden demostrar ("Enunciado de reordenación"). Esto quiere decir que el conjunto $\mathcal{A}$ crece en su cardinalidad (*demostrables*), haciendo que ahora ϕ tenga un valor ϕ_2 mayor que ϕ_1. Lo que significa que $\mathcal{A}$ es más parecido a $\mathcal{C}$, pero no igual.

Si este proceso se continúa de la manera anterior, se llegará a tener que $\mathcal{B}$ será más parecido a $\mathcal{I}$ y en consecuencia también $\mathcal{A}$ será más parecido con $\mathcal{C}$, o sea que cuando $\psi \rightarrow 1$, también $\phi \rightarrow 1$.

Por otra parte, si ahora ponemos al sistema a resolver problemas, es decir, si hacemos crecer el número ϕ, primero sin hacer una reordenación de metas, y en caso de no poder resolverlos, inmediatamente lo intentamos con una reordenación mediante una función de jerarquía $Q(x)$, en caso de volver a fallar hay una inconsistencia en la descripción de la meta, que no está tipificada todavía mediante los predicados "imposs" del sistema. Entonces procedemos a localizar la frase, dentro de la meta, que está causando la inconsistencia. Con ello será posible declararla mediante los predicados "imposs", con lo que se consigue que ψ aumente.

Este proceso se aplica tantas veces como sea necesario para eliminar las inconsistencias ($\psi \rightarrow 1$) dentro de las descripciones de los problemas y de esta manera poder resolverlos basados en el *"Enunciado de reordenación"* □

Capítulo 3

PROGRAMAS DE CÓMPUTO Y EXPERIMENTOS

3.1 Programación del Planeador

Como ya indiqué en al capítulo 1, el planeador que usé para investigar los aspectos de consistencia, completitud, reordenación de metas, jerarquización heurística, etcétera, fue WARPLAN [Warren74], que es un planeador típico, que maneja metas conjuntivas y que resume todas las características de los planeadores de su clase. Por lo que es un buen sistema para investigar las tales cuestiones.

El funcionamiento detallado de este planeador se encuentra en la referencia [Warren74]. Sin embargo, la versión que implementé en una microcomputadora, tiene incorporados algunos cambios y anexiones que hice para poder manejar la *consistencia_segunda*, definida en el capítulo 3; los predicados para jerarquización de metas y otras utilerías que no estaban considerados en el programa original.

El planeador está escrito en el lenguaje de programación PROLOG, por lo que los conceptos, las ideas, los ejemplos y las discusiones que se dan en este capítulo tendrán la sintaxis y semática de este lenguaje.

Funcionamiento de planeador

WARPLAN es un planeador que trabaja con metas conjuntivas, es decir, un estado se describe como una conjunción de átomos (o metas en el presente contexto) tal como se ha venido manejando en los capítulos anteriores.

Para encontrar un plan $P = [a_1; \ldots; a_k]$ que me lleve de un estado inicial dado S_0, a un estado meta descrito por $G = \{g_1, \ldots, g_n\}$, también dado, el planeador procede como sigue:

a) El planteamiento del problema anterior se hace con el predicado "plans" de la siguiente manera: plans(G, S_0).

b) El planeador considera la lectura y procesamiento de G, de izquierda a derecha, empezando con g_1.

c) Para lograr una meta g_i, va a encontrar la acción o acciones que permitan alcanzar, a partir del estado presente, un estado en donde g_i es verdadera. Para esto procede de tres maneras:

   (a) Pregunta si g_i es una situación que se cumple en cualquier estado. Por ejemplo, para el problema de los bloques, el átomo *clear(piso)*, es verdadero siempre. Esto es consultado por el planeador mediante los predicados "always", que forman parte de la descripción de la consistencia de la base de conocimiento del sistema. También, en esta parte, pregunta si el átomo que está probando, es verdadero en el contexto de PROLOG, por ejemplo, una desigualdad: *not_equal(X,Y)*; $X \,/= Y$, $X <= Y$, etc.

   (b) En caso de que lo anterior no se cumpla, va a ver si la situación descrita por g_i forma parte del estado actual, pues en tal caso ya no tendría que buscar nada.

   (c) La tercera manera es buscando en la base de conocimiento una acción que permita obtener a g_i. Esto lo hace mediante el predicado "*add*", que tiene dos argumentos: *add(X,U)*. Este predicado indica que X se obtiene si se realiza la acción U. En este caso X sería instanciada con g_i.

Hasta este punto prodría pensarse que el problema ha quedado resuelto, pues ya hemos encontrado la acción que se andaba buscando. Sin embargo no es así, pues hay que recordar que la acción U tiene sus precondiciones, las cuales pueden o no existir en el estado presente. Por ello, para lograr esta acción U hay dos métodos para hacerlo:

El primero es el más sencillo y es llamado "método por extensión". En términos generales, consiste en cerciorarse si las precondiciones de U están dadas en el estado actual o si se pueden derivar de éste, sin que se destruyan las metas ya logradas anteriormente y que además *sean consistentes* de acuerdo a la *consistencia-segunda*. Si es así, se declara que la acción U y las posibles acciones que pudieran haber sido generadas para derivar U es o son las que se buscaban.

El otro es llamado "método por inserción" y se utiliza cuando el anterior falla. Este es más interesante, pues ahora el planeador tiene que investigar dentro del plan, que hasta el momento ya tenga logrado, si existe un "hueco" en donde pueda *insertar* la acción U, sin que afecte a las metas ya logradas, de tal forma que pueda lograr a la g_i. Para ello considera el siguiente axioma: "*Si la última acción* **W** *del plan presente no elimina o borra a la meta* g_i, *que se está probando, entonces se puede intentar insertar a la acción* **U** *en algún punto anterior a* **W**. *Teniendo en cuenta ignorar a los hechos generados por* **W** *(postcondiciones de* **W***), que se encuentran en una lista de "hechos protegidos" y también de no ignorar a las precondiciones de* **W**".

Figura 3 Esquema de un sistema de planeación

Si este método también falla, quiere decir entonces que el sistema no puede encontrar una acción para lograr a g_i, lo cual indica que existe una inconsistencia no considerada, mediante el predicado "imposs", dentro de la base de conocimiento.

El predicado "consistent" encargado de revisar que las frases que se están trabajando sean consistentes, lo modifiqué para que pudiera efectuar esta revisión hasta la consistencia_segunda, pues originalmente estaba diseñado para revisar solamente la consistencia_primera. Esta modificación además incluyó la revisión, por parte del planeador, de las relaciones entre objetos que antes no hacía y que ocasionaban problemas al estar resolviendo ciertos problemas de planeación. Estas últimas relaciones eran desigualdades entre átomos, predicados que no son átomos definidos para los estados pero que sí estaban definidos en la base de conocimiento, etc.

El resto de las claúsulas definidas en este planeador son auxiliares y utilerías de los predicados principales involucrados en las descripciones anteriores. Son fácilmente entendibles, para alguien entrenado en programar con PROLOG.

3.2 Estructura de la Base de Conocimiento

La base de conocimiento de un planeador independiente del dominio, tiene una estructura general, que se puede aplicar a cualquier universo de objetos que se quiera y que funciona de la misma manera que en un sistema experto. Es por ello que se pueden considerar diferentes dominios y utilizar el mismo sistema de planeación (programa resolvedor de problemas). Un diagrama global de un sistema de planeación se muestra en la figura 3.2.

La base de conocimiento contiene información sobre:

a) Cuáles son las *acciones* definidas para un dominio particular.

b) Las *precondiciones* de tales acciones.

c) Los *átomos* que pueden usarse para definir los *estados* del dominio.

d) Las declaraciones de *consistencia* para el sistema.

e) La descripción del *estado inicial.*

f) Información complementaria, que es particular y depende de cada dominio.

Acciones y Precondiciones

Para definir las acciones (de un dominio dado) y sus correspondientes precondiciones, se utiliza el predicado "can", que tiene la sintaxis: *can(A,C).* En donde A es la acción que se quiere definir y C es la conjunción de átomos que representa a las precondiciones de la acción A. Por ejemplo, para definir la acción de mover un bloque: *move(A,B,C)*, teniendo como precondiciones que A y C no deben tener nada encima y que A debe estar sobre B y que A no puede ser C, se procede, usando "can", de la siguiente forma:

$$\text{can(move(A,B,C), clear(A) \& clear(C) \& on(A,B) \& A} \neq \text{C)}$$

Átomos para los Estados.

Los átomos que se utilizan para definir los estados (inicial, meta, etc.) se definen con los predicados "*add/delete*". En la sección anterior ya se vió una función del predicado "*add*". Además de tener ese significado, sirve para definir cuáles son las *situaciones* (átomos) que pueden darse en los estados. La sintaxis de "*add*" es *add(X,U)* y significa que la situación X se obtiene mediante la acción U. Por ejemplo para indicar que la situación que describe que el bloque A está colocado encima del bloque B: *on(A,B)*, se puede obtener si se mueve el bloque A, de donde se encuentre, a estar encima del bloque B. Esto último es precisamente la acción: *move(A,X,B).* Por lo tanto, esta situación se definiría en la base de conocimiento de la siguiente manera:

$$\text{add(on(A,B), move(A,\_,B))}$$

Para cada declaración del predicado "*add*" debe existir también una declaración correspondiente con el predicado "*delete*". Este predicado indica la forma en la que X deja de ser cierto en un estado determinado. Esto se hace con la ejecución de la acción U. La sintaxis de "delete" es igual a la de "*add*": "*delete*". Un ejemplo sería que, para eliminar la situación que indica que el bloque C está libre, entonces será necesario colocarle un bloque encima. Por ello una definición sería:

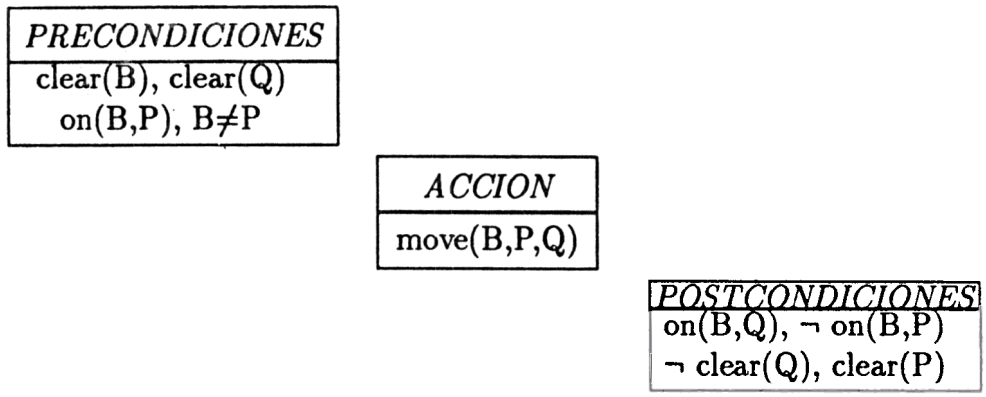


Figura 3.2: Precondiciones y postcondiciones de la acción *move*

delete (clear(X), move(A,_,X))

Esto predicado es utilizado junto con "*add*" para establecer las postcondiciones de una acción, dado que es necesario eliminar y anexar situaciones después de ejecutada una acción. Por ejemplo ver figura 3.2, en al caso de la acción *move(B,P,Q)*, las precondiciones dadas por el predicado "*can*" son: clear(B) & clear(Q) & on(B,P) & $B \neq P$. Después de ejecutada la acción, el mundo ha cambiado en lo siguiente: Las situaciones *clear(Q)* y *on(B,P)* ya no existen, esto es, hay que eliminarlas. En cambio han aparecido nuevas situaciones que son: *on(B,Q)* y *clear(P)*, las cuales hay que añadir a la descripción del mundo actual.

Consistencia

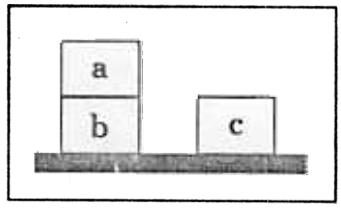
Como ya mencioné en el capítulo 2, los predicados para definir la consistencia del sistema son "*imposs*" y "*always*". Los argumentos para ambos predicados son frases que indican, de manera genérica, los tipos de conjunciones ilegales de átomos, en el caso de "*imposs*". Y cuales son siempre verdaderas para cualquier estado, en el caso de "*always*".

Por ejemplo, para indicar que un bloque no puede estar sobre sí mismo, se coloca en la base de conocimiento la clausula:

imposs(on(X,X))

Para indicar que el piso siempre está libre para colocar encima de él algo, se define en la base de conocimiento la clausula:

always(clear(piso))



Descripción :
given(inicial,on(a,b)).
given(inicial,on(b,piso)).
given(inicial,on(c,piso)).
given(inicial,clear(a)).
given(inicial,clear(c)).

Figura 3.3: Un estado **inicial**

Estado inicial

El estado inicial es la descripción del mundo, a partir de la cual el planeador inicia un proceso. Pueden existir diversas descripciones: $\{S_1, \ldots, S_k\}$ que pueden servir como estados iniciales para distintos procesos. Estos S_i son los que deben darse como segundo argumento del predicado principal $plans(C, S_i)$.

Un estado inicial se especifica mediante el predicado "given", que tiene dos argumentos: El primero es el nombre del estado que se está definiendo (algún S_i). El segundo es *un átomo* que corresponde a cada una de las situaciones que existen dentro del estado en cuestión. Por ejemplo para describir el estado representado en la figura 3.2, se escribiría lo que se muestra a la derecha de la misma figura.

Información complementaria

Dentro de la base de conocimiento, además de la información ya descrita anteriormente, es posible tener otras definiciones adicionales, que dependen exclusivamente de las características del dominio de que se trate. Por ejemplo para el problema de los bloques se tiene la definición, adicional, del predicado *sobre*, que es verdadero si un bloque X se encuentra en la misma pila de bloques en donde se halla otro bloque Y, pero en alguna posición arriba, no necesariamente inmediatamente encima de Y. La definición de este predicado está en el capítulo 3 (sección 3.3, "*El Problema de los bloques*"). Este predicado se escribiría en la base de conocimiento como:

```
sobre(X,Y,U):-elem(on(X,Y),U),!.
sobre(X,Y,U):-elem(on(X,Z),U),sobre(Z,Y,U).
```

En este caso U es la conjunción de átomos que representa a cada una de las situaciones que describen una pila de bloques.

El predicado *elem(X,C)* es una utilería que cuando se usa toma valor verdadero si el átomo X es un elemento de la conjunción C y resulta falso en caso contrario.

(a) (b)

Figura 3.4: Estado inicial Estado meta

3.3 Un ejemplo de planeación

En esta sección quiero presentar, sin entrar en mucho detalle, un ejemplo relativamente sencillo de cómo trabaja el planeador para cuando tiene que resolver un problema.

Nuevamente, escojo el problema de los bloques por ser un dominio claro y sencillo en sus reglas de funcionamiento. Vamos a considerar que tenemos el estado inicial que se muestra en la figura 3.3 a y queremos que el planeador encuentre una solución para obtener el estado que se muestra en la figura 3.3 b.

Descripción del problema genérico

El estado inicial está descrito por los enunciados "*given*" que se encuentran en la figura 3.2 (nótese que es el mismo estado inicial).

Además de la descripción del estado inicial, la base de conocimiento contiene la información global del dominio para el problema de los bloques y que se trató en la sección 3.2, "*Estructura de la Base de Conocimiento*", está dada por los siguientes encunciados:

% Predicados "add/delete" para las situaciones de los *estados*:

```
add(on(U,W),  move(U,V,W)).
add(clear(V), move(U,V,W)).
delete(on(U,W),  move(U,W,V))
delete(clear(W), move(U,V,W))
```

% Definición de las *acciones* y sus *precondiciones*:

```
can( move(U,V,piso), on(U,V) & not_equal(V,piso) & clear(U))
can( move(U,V,W), clear(W) & on(U,V) & not_equal(W,V) &
                  not_equal(U,W) & clear(U)).
```

% Predicados "*imposs*" para la *consistencia*:

```
imposs(on(X,Y) & clear(Y)).
imposs(on(X,Y) & on(X,Z)  &  Y  /=  Z).
imposs(on(X,Y) & on(Z,Y) & X /=  Z).
imposs(on(X,X)).
imposs(on(X,Y) & on(Y,Z) & on(Z,X)).imposs(on(X,Y) & C):-
          sobre(Z,X,C),sobre(Y,Z,C).
```

% Información adicional

```
sobre(X,Y,U):-elem(on(X,Y),U),!.
sobre(X,Y,U):-elem(on(X,Z),U),sobre(Z,Y,U).
```

Planteamiento del problema

Para resolver un problema de planeación, primero, es necesario que el estado meta sea descrito por una conjunción de átomos válidos, esto es, de aquellos que se encuentran en las definiciones "*add/delete*". En este caso, el estado meta debe ser descrito mediante átomos del tipo *clear(X)* y *on(X,Y)*. Así, una descripción del estado en la figura 3.3 b, puede ser la siguiente:

on(a,b) & on(b,c)

Para indicarle al planeador que resuelva este problema, hay que decírselo con el predicado principal *plans* de la siguiente forma:

plans(on(a,b) & on(b,c), inicial)

Con esto el planeador inicia e proceso para encontrar una solución al problema
Traza de la solución.

El programa va a resolver el problema, solucionando una por una las metas que forman el estado meta, empezando por la que se encuentra en el extremo izquierdo y yendo hacia la derecha. En este caso la primera meta por solucionar es **on(a,b)**.

Vamos a considerar dos listas importantes en la solución de los problemas de planeación. Una es para colocar los *hechos protegidos*, es decir aquellos que forman el conjunto de post-condiciones de cada una de las acciones que se vayan generando al resolver el problema. La otra lista es en donde se va colocando o construyendo la solución al problema, esto es, un plan parcial. Llamemos a cada una de estas listas H y P respectivamente. Estas listas constituyen la parte principal del *contexto* (ver figura 3.2).

El programa primero trata de encontrar un plan parcial para lograr la meta "on(a,b)". Pero como esta situación ya está dada en el estado inicial (ver la descripción), lo único que hace es copiar este hecho en la lista H.

Luego toma la siguiente meta que es **on(b,c)**. Este átomo no se encuentra en el estado inicial. Entonces va a tratar de resolverlo. Para ello, consulta en la base de conocimiento (**BC**) la manera de cómo anexar al contexto el hecho "**on(b,c)**".

"*add*" dice que para hacerlo se necesita la acción **move(b,_33,c)**[1]. Pero esto implica que tiene que destruir uno de los hechos protegidos (el único): "on(a,b)". Entonces la estrategia es elaborar un subplan que produzca "on(a,b)" a partir del estado inicial. Esto parece una redundancia, pero es necesaria para el proceso posterior de lograr "on(b,c)".

Este subplan se consigue únicamente utilizando el método por extensión de metas, que no tiene mucha complicación. El subplan consistente de las acciones:

```
inicial;
move(a,b,piso);
move(a,piso,b).
```

Como se puede ver, el resultado final de este subplan es nulo, no altera en nada lo que se tiene.

Ahora ya es posible resolver la meta pendiente: "on(a,c)". El contenido de H sigue siendo "on(a,b)" y el contenido de P es el propio subplan anterior.

La manera de lograr "on(b,c)", es con el método por inserción. Este puede funcionar ahora, porque existe ya un plan parcial (el que está en P) y que anteriormente no se tenía. Para lograr "on(b,c)" se necesita la acción: `move(b,_223,c)`. Ahora, se aplica el axioma de inserción: La última acción, `move(a,piso,b)`, no elimina a la acción que se está probando, es decir, el mover el bloque a del piso a encima del bloque b, no afecta la situación creada por la acción `move(b,_223,c)`, por lo tanto esta última puede ser insertada antes de la otra acción:

```
=> move(b,_223,c) => move(a,piso,b)
```

El programa encuentra que la instancia para la variable anónima _223 es *piso* (porque b está sobre el piso), por lo tanto la expresión anterior, con la variable instanciada y ya unida a la primera parte del plan parcial, queda finalmente como el plan que se buscaba, pues ya no hay más metas que resolver:

```
inicial => move(a,b,piso) => move(b,piso,c  => move(a,piso,b)
```

Este plan es escrito con un formato de salida especial como sigue

```
<< P L A N >>
  inicial;
  move(a,b,piso);
  move(b,piso,c);
  move(a,piso,b).
```

[1]Usaré la notación de PROLOG para las variables anónimas. Así, _# (# es un ordinal), representa a una variable anónima

En este ejemplo para resolver un estado meta no se utilizó la técnica de reordenación por jerarquización de metas, dado que la intención era la de mostrar cómo funciona el planeador únicamente. En la siguiente sección muestro las comparaciones entre las corridas con y sin la reordenación.

3.4 Experimentos desarrollados

El objetivo de esta sección es el de mostrar los resultados de la computadora, obtenidos al aplicar las técnicas descritas en los capítulos precedentes.

Para probar las ideas de reordenación por jerarquización de metas, voy a trabajar con los dos dominios que he venido discutiendo a lo largo de este trabajo: *El problema de los bloques* y *El mundo de STRIPS.*

La jerarquización de metas es un proceso algorítmico que se puede automatizar hasta cierto punto.

De los dos ejemplos tratados en el capítulo 1 (sección 1.3) se puede ver que debido a las heurísticas muy particulares de cada dominio, es difícil construir un proceso automático general para ordenar las metas. Cada dominio tiene sus propias consideraciones heurísticas, que no son iguales a los de otro y por lo tanto un programa general de reordenación tendría que ser muy elemental, o extremadamente complejo, para poder así manejar cualquier dominio.

Por ello, para el problema de los bloques desarrollé unos predicados, escritos en PROLOG, para ordenar las metas jerárquicamente de acuerdo a como se describió en la sección 2.3. El predicado principal es *reordena(C,R)*, en donde el argumento C es la conjunción de metas que se desea reordenar. Y el segundo argumento, R, es la salida del predicado y representa a la conjunción de metas pero ya ordenada de acuerdo a la jerarquización discutida en la sección 2.3. La programación del predicado se encuentra en el apéndice A, junto con todas las clausulas que le sirven de utilería.

Este predicado es llamado antes de entrar al predicado principal de planeación: *plans.* Así, la secuencia de llamado sería:

reordena (C,R), plans(R,S)

en donde S es el nombre del estado inicial, C es el estado-meta y R es C pero ya ordenada.

Para hacer las pruebas de los reordenamientos con el problema de los bloques, construí un predicado que me permitiera probar todas las combinaciones posibles para colocar todos los bloques del problema en una pila (columna) de bloques. Este predicado se llama "pruebas" y tiene un único argumento, que es una lista cuyos elementos son los nombres de los bloques que participan en el problema. Así por ejemplo, para un problema que tuviera 5 bloques definidos en el mundo, ver figura 2.6, se ejecutaría este predicado así: `pruebas([a,b,c,d,e])` . Con ello se iniciaría un proceso de generar todas las combinaciones de átomos del tipo on(X,Y), con cada uno de los 5 elementos de la lista. En este caso se generarían 120 estados-meta, que serían consistentes, pero no todos tendrían una solución, a menos que se reordenaran. En

general, para un problema con n bloques hay $n!$ estados-meta del tipo columna, esto es, en donde todos los bloques se colocan en una sola columna, a partir de un estado inicial dado.

Elaboré pruebas para un mundo con 3 bloques, luego otra para con 4 y finalmente otra con 5. Encontrando los siguientes resultados:

1. Utilizando la función de jerarquía Q(x), se resuelven *todos* los casos singulares, es decir, los casos que sin ordenar son del tipo 2 y 3 (sección 1.1).

2. Si el proceso de planeación para resolver una meta reordenada, de acuerdo a la función de jerarquía Q(x), requiere de k pasos, sin reordenar requeriría del $O(k^2)$ pasos, suponiendo que en este último caso se pueda encontrar una solución, pues podría tratarse de un caso singular.

3. Aproximadamente un 17% de los problemas generados, para cada uno de los mundos, no se pueden resolver con la reordenación. Sino que el sistema los resuelve muy bien sin utilizar la función de jerarquización.

Las Anomalías.

De las observaciones que lleve a cabo mediante los diferentes experimentos realizados con la computadora, encontré que hay dos situaciones por las cuales no puede resolver los casos singulares y son:

El primer problema con el que se enfrenta un planeador para resolver una meta, es la carencia de una suficiente y correcta información sobre la consistencia del mundo que va a manejar. Es claro que este problema es de origen humano, ya que si el diseñador de la base no da la suficiente y apropiada definición de los predicados "imposs", el mundo llegará a tener muchas inconsistencias, que no se pueden ver desde un principio del diseño de la base de conocimiento, y que aparecen durante la demostración de fórmulas por parte del planeador.

Con esta situación, el sistema puede entrar en ciclos infinitos tratando de demostrar alguna fórmula, careciendo de información sobre lo que no se puede hacer. Por lo tanto este problema se resuelve haciendo más consistente la base de conocimiento. Y esto, cada vez que se detecte que el origen de un problema dado es por inconsistencia. Lo cual se puede hacer utilizando la función de jerarquía Q(x), pues si no puede resolver un problema sin reordenar y si tampoco lo puede hacer reordenándolo, entonces se tiene un caso de *inconsistencia_tercera* en la meta.

El planeador, tal y como se encuentra ahora, hace revisiones de las dos primeras consistencias, tanto al recibir del usuario la descripción de la meta, como durante el proceso de planeación.

2. El segundo problema es debido al propio mecanismo deductivo del sistema de planeación, pues en los casos en los que no puede encontrar una solución a un problema planteado, lo que está sucediendo es que para demostrar (lograr) un átomo (una situación) de la frase (estado) que describe la meta, se regresa, como parte del proceso de instanciación de variables, haciendo "backtracking" y a partir de ese punto se plantea el resolver un sub-problema, que resolvería el original, pero no se da cuenta que este sub-problema es el mismo problema original. Y por ende al llegar al mismo punto que en el paso anterior, se regresa igualmente y vuelve a plantearse el mismo problema como un sub-problema, lléndose así hasta el infinito. Claro está que de esta manera, primero agotará toda la memoria disponible en la computadora.

   Esta anomalía, creo que puede resolverse si se amplía la definición de alguna manera del axioma que sustenta al método para lograr una meta por inserción y que dice: "Si la última acción W del plan presente (plan parcial) no elimina la meta g_i que se está probando, entonces se puede intentar insertar la acción U en algún punto anterior a W. Teniendo en cuenta a los hechos que hay hasta antes de W ".

   Sin embargo, puedo decir que, si el problema no se debe a una falta de definición de las inconsistencias en la base de conocimiento (problema 1), entonces la meta es demostrable si se reordena mediante una función de jerarquía $Q(x)$.

El Caso de STRIPS.

El problema del robot de STRIPS me condujo a considerar la "optimalidad" de la reordenación. A diferencia del *problema de los bloques*, con STRIPS no puede resolver muchos de los problemas planteados a no ser que éstos fueran reordenados. Sin embargo, de los problemas que no se reordenaron y que se obtuvo una solución, mostraron que el proceso de planeación requería de un menor número de pasos que cuando se tenían reordenadas las metas. Un ejemplo es el siguiente: Supongamos que queremos el plan para colocar las cajas 2 y 3 juntas y encender la luz del cuarto 1 (ver figura 1.3). Una descripción de la meta para este problema sería la siguiente: `nextto(box(2), box(3)) & status(lightswitch(1), on)`. La orden para elaborar el plan es:

```
plans(nextto(box(2),box(3)) & status(lightswitch(1),on),inicial
```

La respuesta del sistema de planeación:

```
El plan original es:
nextto(box(2), box(3)) & status(lightswitch(1), on)


    <<P L A N >>
  inicial ;
  goto2(box(2), room(1)) ;
  pushto(box(2), box(3), room(1)) ;
  goto2(box(1), room(1)) ;
  pushto(box(1), lightswitch(1), room(1)) ;
  climbon(box(1)) ;
  turnon(lightswitch(1)).

Se utilizaron 8 pasos
```

Si ahora reordenamos los átomos de la meta dada, de acuerdo a la función de jerarquía de este problema (ver capítulo 2, sección 2.3), tenemos que la orden para solicitar el plan es:

```
plans(status(lightswitch(1),on) & nextto(box(2),box(3) ,inicial
```

Y la respuesta del planeador es

```
El plan original es:

 status(lightswitch(1), on) & nextto(box(2), box(3))

    << P L A N >>
  inicial ;
  goto2(box(1), room(1)) ;
  pushto(box(1), lightswitch(1), room(1))
  climbon(box(1)) ;
  turnon(lightswitch(1)) ;
  climboff(box(1)) ;
  goto2(box(2), room(1)) ;
  pushto(box(2), box(3), room(1)).

Se utilizaron 9 pasos
```

En la parte inferior del recuadro que contiene la secuencia de acciones del plan producido, se da el número de pasos utilizados durante el proceso de planeación para resolver la meta planteada.

Como se puede ver, el número de pasos requeridos para resolver la meta sin reordenar es menor que reordenándola. Pero si ponemos atención a la secuencia producida por un plan y otro, observamos que en el primer caso la primera meta que debe ejecutar el robot es colocar la caja 2 junto a la caja 3 (" `nextto(box(2),box(3))`"). Luego debe ir a la caja 1 y empujarla hasta la posición en donde se encuentra el interruptor de la luz. Se sube a la caja y enciende el interruptor. Y ahí termina su tarea.

En cambio en el segundo caso, primero tiene que empujar la caja 1 hasta el interruptor de luz, subirse, encenderlo, *bajarse* y luego ir a juntar las cajas 2 y 3.

La diferencia entre un plan y otro, está en la acción adicional de *bajarse* de la caja 1 (en el segundo caso). La cual es muy lógica e inevitable y forma parte, además, de la consistencia del mundo de STRIPS.

3.5 Utilización del sistema

Esta sección constituye un breve manual de usuario del sistema de planeación "WARPLAN modificado", que puede correr en cualquier microcomputadora que tenga el leguaje PROLOG. Sería muy conveniente utilizar una versión de PROLOG que siguiera lo más fielmente posible la sintaxis de Clocksin-Mellish, aunque no es indispensable, pues siempre es posible hacer adaptaciones o equivalencias en otras versiones con sintaxis diferentes.

De hecho, con los ejemplos que he venido desarrollando en el transcurso de los diferentes capítulos, uno se puede dar cuenta aproximada, de cómo es la forma de utilizar el planeador. Por ello, en esta sección solo voy a resumir todo lo relacionado a cómo usarlo.

El Resolvedor de Problemas y la Base de Conocimiento del planeador están escritos totalmente en PROLOG, usando la sintaxis de Clocksin-Mellish. El contexto es construído por el resolvedor de problemas durante el proceso de demostrar una fórmula (frase). Y solamente el resolvedor tiene acceso a él. Después de terminado el proceso, se pierde.

El contenido de la base de conocimiento está claramente descrito en la sección anterior de este capítulo y puede ser creada, modificada, etc., utilizando cualquier editor de textos convencional.

Dado que la base de conocimiento es un archivo para PROLOG, el orden en el cual son escritas las clausulas que tienen el mismo encabezado, es importante. Por ello el usuario debe considerar, al momento de diseñar sus predicados "can", el orden en el cual quiere que sean considerados. Por ejemplo en el caso de los bloques, me interesa que cuando vaya a efectuar el movimiento de un bloque, primero considere colocarlo en el piso, que moverlo hacia una posición encima de otro bloque. Esto tiene dos razones: Si lo coloca sobre otro bloque, puede entonces, estorbar el movimiento del bloque que se encuentra debajo de él. Y otra es que, colocados los bloques en el piso, se logra tener un "*estado canónico*", en el que todos o la mayoría de los bloques se encuentran sobre el piso y entonces el formar pilas de bloques a partir de este estado canónico es un proceso sumamente sencillo.

Como se ve de lo anterior, es posible y hasta necesario colocar en la base de conocimiento información heurísti ca, que no puede ser puesta en otra parte del sistema, sino mediante un

"*orden heurístico*" dentro de la propia base. Es por ello que el mantenimiento de la base de conocimiento debe realizarlo una persona especializada que conozca bien su estructura.

La descripción del estado inicial es dada mediante las clausulas "*given*", las cuales pueden ir dentro de la base de conocimiento o en cualquier otra parte. Dado que es variable esta descripción, muy bien puede leerse de un archivo que sólo la contenga y así tener tantas descripciones como se quiera. La forma de este predicado ya se vió en la sección anterior, con un ejemplo.

La descripción del estado inicial también es muy importante, porque es la descripción misma del mundo a partir de donde se va a buscar la solución. Una descripción pobre puede producir resultados no deseados, en el mejor de los casos, y en el peor, el planeador entraría en ciclos infinitos al tratar de encontrar una solución careciendo de mucha información necesaria para ello.

Entre más rica sea la información que describe un estado inicial, el planeador tendrá más oportunidades de encontrar soluciones a los problemas planteados. No le afecta el contar con una gran cantidad de información, salvo en la cantidad de memoria que consuman las descripciones y en el tiempo de consulta a esta información.

Tal vez a una persona le parezcan triviales algunas situaciones del mundo, inclusive las puede considerar como obvias por ser del sentido común, pero hay que recordar que el sistema de cómputo no las sabe y no se pueden hacer supuestos de que las *conoce*. Aun los humanos, con su sentido común pueden discrepar uno de otro, si se les pide que construyan, cada uno por separado, una versión del mundo que se les describe, verbalmente, mediante situaciones que forman tal estado del mundo. Al final si vemos los resultados, nos sorprenderá la diversidad de versiones construídas por cada persona. Muy difícilmente se dará que dos o más coincidan exactamente en sus resultados. Así pues, la descripción precisa, completa y detallada del mundo es fundamental, tanto para el sistema como para el usuario.

La descripción de un estado meta se da mediante una frase, que deberá tener todas y cada una de las situaciones que se desea tener en el estado final. Es claro que si algunas de las situaciones del estado inicial van a prevalecer hasta el final, entonces no es necesario incluirlas en la descripción del estado meta.

Las frases son construídas *únicamente* con los átomos que se definen en los predicados "*add/delete*". El orden en el cual son escritos es el tema central de este trabajo. Normalmente no importaría el orden en el cual fuesen escritos. Pero no es así, pues para resolver los casos de singularidad es necesario reordenar con la función de jerarquía $Q(x)$.

Si suponemos que el predicado "*plans*" utiliza un predicado para la reordenación de metas (*reordena*) entonces podemos proporcionarle la frase del estado meta y el estado inicial S_0 de la siguiente forma: plans(frase_meta, S_0). Después de esto el sistema empieza la búsqueda de una solución para la frase_meta. También el predicado "reordena" puede estar fuera de "*plans*" y ser llamado justo antes de éste. Con ello se tendría la opción de reordenar o de no hacerlo antes de planear, y considero que esto sería lo más conveniente.

Capítulo 4

CONCLUSIONES

En este capítulo resumo las características, alcances, limitaciones y aportaciones de mi trabajo de investigación.

Formalización.

Con respecto a la formalización, es necesario decir que es una tarea difícil y delicada, pero su objetivo primordial es el de mostrar tanto la estructura como las funciones de un sistema, con toda claridad, tal como lo hacen los planos de una máquina. Una vez que queda formalizado un sistema deductivo, las relaciones lógicas entre las proposiciones matemáticas quedan prácticamente a la vista, permitiendo con ello ver las estructuras de configuraciones de ciertas cadenas de símbolos, sin aparente significación, como se van enlazando unas con otras, como se combinan sintácticamente, como penetran unas en otras, etc.

Una de las constribuciones de este trabajo ha sido la presentación de una formalización de la Planeación, la cual me permitió desarrollar, bajo un formalismo matemático, las ideas de consistencia y completitud de los sistemas de planeación, las cuales sin esta base serían difíciles de establecer con claridad.

Con este enfoque vimos que hay una serie de problemas teóricos y prácticos que se pueden investigar, pues aun este campo no ha sido muy explorado. Más adelante menciono algunos de ellos.

Reordenación

La reordenación de metas, fue la clave principal para el desarrollo de este trabajo. Esto no fue el producto de un proceso deductivo normal, sino que fue un hallazgo que ocurrió, como muchos otros descubrimientos de la investigación científica y/o tecnológica, durante una etapa de pruebas y experimentos, al estar trabajando con los sistemas de planeación.

El algoritmo de reordenación se basa en la idea de que para resolver un problema dado, hay que empezar a resolver primero, las partes más complejas e ir avanzado hacia las más

simples. En otras palabras, es resolver primero aquello que no va a estorbar a otras metas más adelante (lo que es precisamente planear). Es como si se quisiera armar un motor de combustión interna y al final, cuando ya estuviera todo cerrado, se quisieran colocar los pistones. Esta estrategia es muy humana e involucra mucho conocimiento heurístico. Por ello el algoritmo de jerarquización está completamente basado en consideraciones heurísticas. Esto es como pasarle nuestro conocimiento, para resolver problemas, a la máquina. Ella sólo tiene un mecanismo para emular, en una pequeña fracción, los procesos deductivos del cerebro humano. Y no tiene conocimiento alguno de cómo resolver un problema. Le podemos decir que hay reglas, procedimientos para tratar los problemas, pero con ello no le estamos dando todo el conocimiento que nosotros usamos para resolverlos. En gran mediada es el sentido común, la experiencia, la inteligencia misma y también la intuición que produce nuevas formas de resolver los problemas. Una parte de estos aspectos es algo de lo que debemos incluir en el "conocimiento" de la máquina.

Con esta jerarquización de metas fue posible replantear los problemas que no tenían solución (casos singulares), logrando con ello que el planeador encontrara soluciones para esos casos.

Otra característica favorable del algoritmo es que al aplicarlo a la mayoria de los problemas que no se pueden resolver mediante la tecnica normal, el proceso de planeación requiere de menos pasos para encontrar la solución. Y en algunos casos, también es más corta con respecto al número de acciones involucradas.

Anomalías.

Encontré dos limitaciones para este algoritmo, las cuales pueden ser eliminadas si se trabaja un poco más en ellas. Pero esto se sale de los alcances de esta investigación. Y se dejan como problemas abiertos.

Ambas limitaciones se trataron en detalle en el capítulo 3 (sección 3.4). Y la primera es que hay problemas que reordenándolos, no pueden ser resueltos por el planeador. Aunque sí los puede resolver sin la reordenación. Este es un problema dentro del mecanismo del Resolvedor de Problemas, que puede muy bien ampliarse en su función mediante una redefinición, primero del "axioma de inserción" y luego en la definición de las propias clausulas "achieve", que son parte del resolvedor de problemas del planeador y que se usan para tratar de lograr una meta.

El segundo problema es que la reordenación puede producir planes con un número mayor de acciones y también de pasos en el proceso de planeación, que cuando se plantean las mismas metas sin reordenar. Esto, como se discutió en la sección 3.4, es algo que no se puede evitar, debido a la secuencia lógica en la que deben producirse las acciones de un plan. Sin embargo, utilizando una mejor jerarquización, tal vez más complicada, sea posible cambiar esta situación. También es un punto para desarrollo posterior.

Consistencia y completitud.

Al aplicar el algoritmo de jerarquización a un mundo determinado, uno se puede dar cuenta de cuales son las inconsistencias que deben de definirse en la base de conocimiento, mediante las clausulas "imposs". Aparte de las que ya se tuvieran definidas con anterioridad, de acuerdo al conocimiento global que se tuviese del dominio en cuestión. Por lo tanto, puedo decir que la jerarquización indirectamente me permite hacer un mejor y más completo diseño de las inconsistencias del mundo.

Para un sistema de planeación y un dominio dados, es posible lograr hacer de ellos un sistema completo, desde el punto de vista matemático. Pues para aquellas fórmulas que no se tenga una demostración, en un momento dado, se pueden reordenar jerárquicamente y obtener, con un 65% de probabilidad, una demostración. Si esto no sucede, entonces hay todavía en el sistema inconsistencias aun no declaradas en la base de conocimiento. Por lo cual será necesario aumentar el conjunto generado por los "imposs", para que consideren la inconsistencia que origina la irresolubilidad de la fórmula en cuestión. De esta manera el coeficiente de demostrabilidad del sistema tiende a 1, haciendo con ello que el sistema sea completo. En el caso de los bloques, definiendo bien las inconsistencias para que cubrieran todos los casos de inconsistencias, logré tener un sistema completo.

Problemas abiertos.

Aprovechando la formalización de los elementos involucrados en la planeación, es posible hacer desarrollos teóricos que antes no era posible hacer debido a una herramienta formal para trabajar estas ideas, pues solo había técnicas, algoritmos, métodos de programación, etc.

Algunos temas de investigación, ya los he señalado anteriormente y otros son los siguientes:

1. Hemos visto que es posible que un sistema de planeación sea completo, matemáticamente, mediante la consideración de heurísticas introducidas en el sistema. Luego, ? estos sistemas de planeación forman un conjunto que pertenece tal vez a un superconjunto de entidades que cumplen con el teorema de completitud de Gödel?

2. Investigar nuevas alternativas para las medidas de los cocientes ψ y ϕ (capítulo 2). Tal vez utilizando probabilidades, con el objeto de medir cuantitativamente los límites de consistencia y completitud de los sistemas de planeación.

3. Investigar una posible clasificación de los problemas que son solubles por los sistemas de planeación, de acuerdo con ciertas "complejidades". Por ejemplo, con las máquinas

de Turing, los problemas se clasifican en cuanto al tiempo que tardan en resolver un problema o en cuanto al espacio requerido en la cinta de la máquina de Turing.

4. Ver la posibilidad de manejar de alguna manera directa o indirectamente (con otra estrategia heurística) la inconsistencia_tercera, pues con ello se aceleraría definición de las inconsistencias de un dominio determinado.

5. La heurística de reordenación, se puede ver como una aproximación al problema del parado de la máquina de Turing. No quiere decir que resuelve ese problema, dado que pertence a un dominio en donde los sistemas son indecidibles, sino que aprovechando el isomorfismo se pueden estudiar estos aspectos pero desde los sistemas de planeación.

Último comentario.

Finalmente quisiera mencionar algo sobre las conclusiones de Kurt Gödel sobre su *Teorema de Incompletitud.* Uno podría preguntarse si es posible construir máquinas computadoras que sustituyan la inteligencia del hombre para hacer matemáticas. Pues considerando las arquitecturas actuales de computadoras secuenciales, teniendo grandes velocidades de cómputo y grandes cantidades de memoria, junto con un "software" como el descrito en este trabajo o aun más sofisticado, son al fin y al cabo un *conjunto finito* de instrucciones. A la luz del teorema de incompletitud de Gödel, existe un conjunto infinito de problemas de la teoría elemental de los números que tales configuraciones complejas de máquinas + software son incapaces de resolver. Aun es muy factible que el propio cerebro humano, considerado como una máquina de cómputo muy sofisticada, pero con limitaciones inherentes a su estructura, no pueda resolver ciertos problemas matemáticos. Sin embargo la estructura bio-físicoquímica del cerebro humano contiene una estructura compleja de reglas operativas mucho más poderosa que la de cualquier computadora existente y aun proyectada. Así pues, no hay una perspectiva razonable que indique que la mente humana sea desplazada por cerebros artificiales, hechos por el hombre mismo.

Esto no debe constituir un desaliento para el desarrollo de las matemáticas ni de la ciencia en general, ni para entregarse a inútiles y dañinas especulaciones misteriosas. Pues el hecho de haber descubierto algunas verdades aritméticas indemostrables no implica que haya verdades eternamente insuceptibles de descubrimiento, ni que la demostración constrictiva deba sustituirse por una intuición mística. Más bien significa que los recursos de la inteligencia humana no han sido ni pueden ser completamente formalizados ni encajonados en modelos construídos por la propia mente del hombre, y que pueden esperarse nuevos principios de demostración para la invención y los descubrimientos. Se ha visto que proposiciones matemáticas que no pueden establecerse mediante una deducción formal a partir de un conjunto dado de axiomas, pueden, sin embargo, establecerse mediante razonamiento

metamatemático "informal". De hecho la propia obra de Gödel es una muestra clara y notable del potencial y sutileza de la estructura del cerebro humano, lo cual una vez más nos da ocasión de apreciar el poder de la razón creadora.

Bibliografía

[Allen83] Allen, J.F. y Koomen, J.A.; *Planning Using a Temporal World Model*; IJCAI-83, pp 741-747, Karlsruhe, Alemania Federal; (1983) [TIMELOGIC].

[Appelt85] Appelt, D.E.; *Planning English Referring Expressions*; Artificial Intelligence, 26, pp 1-33, (1985) [KAMP].

[Barrow75] Barrow, H.G.; *HBASE: A Fast Clean Efficient Data Base System*; D.A.I. POP-2 Library Documentation; Edinburgh University (1975) [HBASE].

[Bobrow76] Bobrow, D.G. y Winograd, T.; *An Overview of KRL - a Knowledge Representation Language*; Xerox PARC Report CSL-76-9, Xerox PARC, Ca, USA, (1976) [KRL].

[Bobrow83] Bobrow, D.G. y Stefik, M.J.; *The LOOPS Reference Manual*; Xerox PARC, Ca., USA; (1983) [LOOPS].

[Bresina81] Bresina, J.L.; *An Interactive Planner That Creates Structured Annotated Trace of its Operation*; Rutgers University, Computer Science Research Laboratory, Reporte CBM-TR-123;(1981) [PLANX-10].

[Chang73] Chang,Ching-Lian y Lee, Richard Char-Tung; *Symbolic Logic and Mechanical Theorem Proving*; Academic Press,Inc., 1973.

[Chang88] Chang Kai-Hsiung y Wee, W.G.; *A Knowledge-Based Planning System for Mechanical Assembly Using Robots* ; IEEE Expert Vol.3 No.1, Spring'88, Engineering/Manufacturing, 1988.

[Chapman87] Chapman, D.;*Planning for Conjuntive Goals*; Artificial Intelligence 32 (1987) pp. 333-377.

[Clark85] Clark, P.E.; *Towards an Improvement Domain Representation for Planning*; Tesis de maestria, Departament of Artificial Intelligence, University of Edinburgh, 1985.

[Clocksin84] Clocksin, W.F. y C.S. Mellish; *Programming in PROLOG*; Segunda edicion, Springer-Verlag, 1984.

[Corkill79] Corkill, D.D.; *Hierarchical Planning in a Distributed Environment*; IJCAI-79, pp 168-175, Tokio, Japon; (1979).

[Corkill83] Corkill, D.D. y Lesser, V.R.; *The Use of Meta-level Control for Coordination in a Distributed Problem Solving Network*;IJCAI-83, pp 748-756, Karlsruhe, Alemania Federal; (1983).

[Daniel83] Daniel, L.; *Planning and Operation Research*; Artificial Intelligence: Tools, Techniques and Applications, Harper and Row, New York, (1983) [NONLIN].

[Davis73] Davis, D.J.M.; *POPLER 1.5 Reference Manual*; D.A.I. Theoretical Psychology Unit Report no. 1, Edinburgh University.(1973) [POPLER].

[Dean85] Dean, T.; *Temporal Reasoning Involving Counterfactuals and Disjuntions*; IJCAI-85, Los Angeles, Ca, USA; (1985) [TNMS].

[Deering81] Deering, M., Faletti, J. y Wilensky, R.; *PEARL: An Efficient Language for Artificial Intelligence Programming*; IJCAI-81,Vancouver, British Columbia, Canada; Agosto, 1981, [PEARL].

[Dershowitz] Dershowitz, N.; *Synthetic Programming*; Artificial Intelligence, 25, pp 323-373.

[Doran66] Doran, J.E. y Michie, D.; *Experiments with the Graph Traverser Program*; Proceedings of the Royal Society, A, pp 235-259 (1966) [GRAPH TRAVERSER].

[Drummond85] Drummond, M.E.; *Refining and Extending the Procedural Net*;IJCAI-85, Los Angeles, Ca. (1985).

[Erman] Erman, L.D., Hayes-Roth, Lesser, V.R., Reddy, D.R.; *The HEARSAY-II Speech-understanding System: Integrating Knowledge to Resolve uncertainty*; ACM Computing Surveys, 12,No. 2; (1980).

[Fahlman79] Fahlman, S.E.; *NETL: a System for Representing Real World Knowledge*; MIT Press, Cambridge Mass., USA; (1979) [NETL].

[Fikes71] Fikes, R.E. y Nilsson, N.J.; *STRIPS: A New Approach to the Application of the Theorem Proving to Problem Solving*; Artificial Intelligence 2, pp 189-208 (1971).

[Fikes72a] Fikes,R.E., Hart, P.E. y Nilsson, N.J.; *Some new directions in robot problem solving*; en: B. Meltzer y D. Michie, (Eds.), Machine Intelligence 7 Edinburgh University Press, Edinburgh, 1972 Cap.23.

[Fikes72b] Fikes, R.E., Hart, P.E. y Nilsson, N.J.; *Learning and Executing Generalized Robot Plans* ; Artificial Intelligence 3,(1972) [STRIPS].

[Fikes82] Fikes, R.E.; *A Commitment-based Framework for Describing Informal Cooperative Work*; Cognitive Science, 6, pp 331-347 (1982).

[Fox81] Fox, M.S., Allen, B. y Strohm, G.; *Job Shop Scheduling: An Investigation in Constraint-based Reasoning*; IJCAI-81, Vancouver, British Columbia, Canada, Agosto 1981 [ISIS-II].

[Fox83] Fox, M.; *SRL User's Manual*; Technical Report, Robotics Institute, CMU, Pittsburgh, Pa., USA(1983) [SRL].

[Georgeff82] Georgeff, M.; *Communication and Interaction in Multi-agent Planning Systems*; AAAI-3, (1982).

[Green69a] Green, C.C.; *Application of Theorem Proving to Problem Solving*; Proceedings IJCAI-69, pp. 219-239 Washington, D.C., Mayo 1969.

[Green69b] Green, C.C.; *Theorem Proving by Resolution as a Basis for Question Answering*; (en Machine Intelligence 4, ed. Meltzer,B. y Michie, D).; Edinburgh University Press (1969).

[Guzman84] Guzmán, A.A.; *AHR : una Arquitectura Heterárquica Reconfigurable*; Reporte Técnico , Dpto. Ingeniería Eléctrica, CINVESTAV (1984). [AHR].

[Hayes-Roth78] Hayes-Roth, B. y Hayes-Roth, F.; *Opportunistic Problem Solving*; en The Handbook of Artificial Intelligence, Vol. III, XI Models of Cognition, C, pp. 22-27 (P.R. Cohen y E.A. Feigenbaum editores), William Kaufmann, Inc.,Los Altos, Calif. (1982).

[Hayes-Roth79] Hayes-Roth, B. y Hayes-Roth, F,; *A Cognitive Model of Planning*; Cognitive Science; pp 275-310; [OPM] (1979).

[Hayes-Roth80] Hayes-Roth, B.; *Human Planning Processes* ; Rep. No.R-2670-ONR, Rand Corp., Santa Monica, CA (1980).

[Hermes69] Hermes,H.; *Enumerability.Decidability computability*; An Introduction to the Theory of Recursive Functions; 2da ed. revisada; Springer-Verlag, 1969.

[Hewitt71] Hewitt,C.; *PLANNER: A Language for Proving Theorems in Robots*; Proc. IJCAI 2, 1971

[Hewitt72] Hewitt, C.; *Description and Theoretical Analysis (using schemata) of PLANNER*; MIT AI Lab. Memo No. MAC-TR-256 (1972)[PLANNER].

[Hofstadter80] Hofstadter,D.R.; *Gödel, Escher, Bach: An eternal golden braid*; Vintage Books, 1980.

[Hopcroft79] Hopcroft, J.E. y Ullman, J.D.; *Introduction to Automata Theory , Languages, and Computation* ; Addison-Wesley Publishing Co., 1979.

[Konolige80] Konolige, K. y Nilsson, N.J.; *Multi-agent Planning Systems*; AAAI-1, pp 138-142, Stanford, Ca., USA; (1980).

[Kornfeld79] Kornfeld, W.A.; *ETHER: A Parallel Problem Solving System*;IJCAI-79, pp 490-492, Tokio, Japon; (1979).

[Kowalski79] Kowalski, R.; *Logic for Problem Solving* ; North-Holland, Artificial Intelligence Series 7, 1979.

[Loveland78] Loveland, D.W.; *Automated Theorem Proving: A Logical Basis*; Fundamental Studies in Computer Science, Vol 6; North-Holland Publishing Co.q, 1978.

[Lansky85] Lansky, A.; *Behavioral Planning for Multi-agent Plans*; SRIAI Center Report, (1985).

[Luckham74] Luckham, D.C. y Buchanan, J.R.; *Automatic Generation of Programs Containing Conditional Statements*; AISB Summer Conference, University of Sussex, UK, pp 102-126, julio (1974).

[Manna80] Manna, Z. y Waldinger R.; *A Deductive Approach to Program Synthesis*; ACM Transactions on Programming Languages and Systems 2, ene,1980, pp. 90-121.

[Manna85] Manna, Z. y Waldinger R.; *The origin of the binary-search paradigm*; Proceedings of the 9th IJCAI, Los Angeles, Ca., Agosto 1985, pp 222-224.

[Manna87] Manna, Z.,Waldinger, R.; *How to Clear a Block: A Theory of Plans*; Journal of Automated Reasoning 3(1987) pp 343-377.

[Masui83] Masui, S., McDermott, J. y Sobel, A.; *Decision Making in Time Critical Situation*; IJCAI-83, pp 233-235; Karlsruhe, Alemania Federal; [AIRPLAN] (1983).

[McCarthy63] McCarthy, J.; *Situations, actions, and causal laws*; rep.tec., Stanford University, Stanford, Ca. 1973. Reimpreso en Semantic Information Processing (M. Minsky, ed.), MIT Press, Cambridge MA, 1968, pp. 410-417.

[McCarthy69] McCarthy, J. y Hayes, P.J.; *Some Philosophical Problems from the Standpoint of Artificial Intelligence*; (Machine Intelligence 4, ed. Meltzer,B. y Michie,D.); Edinburgh University Press (1969).

[McDermott78] McDermott, D.V.; *Planning and Acting*; Cognitive Science, 2; (1978).

[McDermott82] McDermott, D.V.; *A Temporal Logic for Reasoning about Processes and Plans*; Cognitive Science, 6, pp 101-155(1982).

[McGregor77] McGregor, D.R. y Mallon, J.R.; *The FACT Database: A System Using Generic Associative Networks*; research report No. 2/80 Dept. of Computer Science, University of Strathclyde, UK; (1977).

[Mellish84] Mellish, C.S.; *Towards Top-Down Generation of Multi-paragraph Text*; Proceedings of the Sixth European Conference on Artificial Intelligence, pp 229, Pisa, Italia, Septiembre 1984.

[Michalski80] Michalski, R.; *Pattern Recognition as Rule-Guided Inductive Inference*; IEEE Transactions on PAMI, Vol. PAMI-2, No.4, Julio de 1980, pp. 349-361.

[Moore80] Moore, R.; *Reasoning about Knowledge and Action*; SRI AI Center Report No. 191, (1980).

[Newell63] Newell, A. y Simon, H.A.; *GPS: a Program that Simulates Human Thought*; (en Feigenbaum, E.A. & Feldman, J. eds.), "Computers and Thought", McGraw-Hill, New York, 1963, [GPS].

[Rulifson Rulifson, J.F., Derkson,J.K. y Waldinger,R.J.; *QA4: A Procedural Calculus for Intuitive Reasoning*; Technical Note 73,SRI International Menlo Park, Ca., USA. [QA4].

[Sacerdoti73] Sacerdoti, E.D.; *Planning in a Hierarchy of Abstraction Spaces*; Artifical Intelligence 5(2) pp 115-135, (1974) [ABSTRIPS].

[Sacerdoti76] Sacerdoti, E.D., Fikes, R.E., Reboh, R., Sagalowicz, D. y Waldinger, R.J.; *QLISP:A Language for the Interactive Development of Complex Systems*; SRI Technical Note, SRI International AI Center, Stanford, Ca., USA; (1976) [QLISP].

[Sacerdoti77] Sacerdoti, E.D.; *A Structure for Plans and Behaviour*; Elsevier-North Holland (1977) [NOAH].

[Saldana87] Saldaña-Aldana, H.; *Técnicas de los Sistemas de Planeación Basados en Conocimiento* ; Reporte Técnico , Serie Amarilla, Dpto. Ingeniería Eléctrica, CINVESTAV, (1987).

[Sathi85] Sathi,A., Fox,M.S. y Greenberg, M.; *Representation of Activity Knowledge for Planning Management*; numero especial de la revista IEEE Transactions on PAMI, julio 1985 [CALLISTO].

[Schank77] Schank, R.C. y Abelson, R.P.; *Scripts, Plans, Goals and Understanding* ;Lawrence Erlbaum Press, Hillsdale, NewJersey, USA (1977).

[Siklossy73] Siklossy, L. y Dreussi, J.; *An Efficient Robot Planner that Generates its Own Procedures*; IJCAI-73, Palo Alto, Ca, USA (1973) [LAWALY].

[Sloman83] Sloman, A.; *POPLOG - a Multi-purpose, Multi language Program Development Environment*; Cognitive Studies Program, University of Sussex, UK. (1983) [POPLOG].

[Smith77] Smith, R.G.; *The Contract Net: a Formalism for the Control of Distributed Problem Solving*; IJCAI-77, pp 472, Cambridge,Mass., USA, (1977).

[Stallman77] Stallman, R.M. y Sussman, G.J.; *Forward Reasoning and Dependency directed Backtracking*; Artificial Intelligence,9, pp 135-196, (1977).

[Stefik79] Stefik, M.J.; *An Examination of a Frame-Structured Representation System* ; IJCAI-79, pp 845-852, Tokio, Japon; (1979)[UNITS].

[Stefik81a] Stefik, M.J.; *Planning with Constraints*; Artificial Intelligence, 16, pp 111-140 [MOLGEN] (1981).

[Stefik81b] Stefik, M.J.; *Planning and Meta-planning*; Artificial Intelligence, 16, pp 141-169, (1981) [MOLGEN].

[Sussman72] Sussman, G.A. y McDermott, D.V.; *Why Conniving is Better than Planning*; MIT AI Lab. Memo 255A; (1972) [CONNIVER].

[Sussman73] Sussman, G.A.; *A Computational Model for Skill Acquisition*; MIT AI Lab. Memo no. AI-TR-297 (1973).

[Tate74] Tate, A.; *INTERPLAN: A plan generation system which can deal with interactions between goals*; Machine Learning Research Unit, Memo MIP-R-109, University of Edinburgh, Edinburgh, 1976.

[Tate75] Tate, A.; *Interacting Goals and Their Use*; IJCAI-75,215- 218, Tbilisi, URSS; (1975). [INTERPLAN].

[Tate76] Tate, A.; *Project Planning Using a Hierarchical Non-linear Planner*; DAI, Reporte 25,Edinburgh University; (1976) [NONLIN].

[Tate77] Tate, A.; *Generating Project Networks*; IJCAI-77, Boston, MA,USA; (1977) [NONLIN].

[Tate84a] Tate, A.; *Planning and Condition Monitoring in a FMS*;International Conference on Flexible Automation Systems; IEE, London, UK, Julio 1984 [NONLIN].

[Tate84b] Tate, A. y Whiter, A.M.; *Planning with Multiple Resource Constraints and an Application to a Naval Planning Problem*; First Conference on the Application of AI; Denver CO.,USA; Diciembre 1984 [NONLIN].

[Tate84c] Tate, A.; *Goal Structure: Capturing the Intent of Plans*; Proc. of the 6th European Conference on AI, Pisa, Italia, Sept. 1984 [NONLIN].

[Tate85] Tate, A.; *A Review of Knowledge Based Planning Techniques*; KBPG-AIAI, University of Edinburgh 1985.

[Vere83] Vere, S.; *Planning in Time: Window and Duration for Activities and Goals*; IEEE Transactions on PAMI-5, No. 3, pp 246-267, Mayo 1983, [DEVISER].

[Waldinger75] Waldinger, R.; *Achieving Several Goals Simultaneously*; SRIAI Center Technical Note 107, SRI, Menlo Park, Ca., USA(1975).

[Warren74] Warren, D.H.D.; *WARPLAN: a system for generating plans*; DCL Memo 76, AI, Edinburgh University (1974).

[Warren76] Warren, D.H.D.; *Generating Conditional Plans and Programs*; Proceedings of the AISB Summer Conference, pp 344-354,University of Edinburgh, UK, Julio (1976).

[Wilensky] Wilensky, R.; *Understanding Goal-based Stories*; Dept. of Computer Science, Yale University, Research Report No.140.

[Wilensky81] Wilensky, R.; *Meta-planning: Representing and Using Knowledge about Planning in Problem Solving and Natural Language Understanding*; Cognitive Science 5, pp 197-233; (1981).

[Wilkins83] Wilkins, D.E.; *Representation in a Domain Independent Planner*; IJCAI-83, pp 733-740, Karlsruhe, Alemania Occidental; [SIPE] (1983).

El jurado designado por la Sección de Computación del
Departamento de Ingeniería Eléctrica del
Centro de Investigación y Estudios Avanzados del
Instituto Politécnico Nacional,
aprobó esta Tesis el 25 de septiembre de 1989.

Dr. Manuel E. Guzmán Rentería.

Dr. José L. Gordillo Moscoso.

Dr. René Bañares Alcántara

Dra. Cristina Loyo Varela

Dr. Guillermo B. Morales Luna