UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

# SISTEMAS DE BASES DE DATOS BASADOS EN LA LÓGICA

TESIS
PRESENTADA A LA JUNTA DIRECTIVA
DE LA FACULTADA DE INGENIERÍA
POR

**BAYRON WOSVELY LÓPEZ LÓPEZ**

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, julio DE 1996

PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

03
T(37-29)
ej. 4

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de tesis titulado:

**SISTEMAS DE BASES DE DATOS BASADOS EN LA LÓGICA**

tema que me fuera asignado por la Coordinación de la carrera de Ingeniería en ciencias y sistemas, con fecha 16 de marzo de 1,995.

Bayron Wosvely López López

PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

MIEMBROS DE LA JUNTA DIRECTIVA

| | |
|---|---|
| DECANO | Ing. Julio Ismael González Podszueck |
| VOCAL 1o. | Ing. Miguel Angel Sánchez Guerra |
| VOCAL 2o. | Ing. Jack Douglas Ibarra Solórzano |
| VOCAL 3o. | Ing. Juan Adolfo Echeverria Méndez |
| VOCAL 4o. | Br. Fernando Waldemar De León Contreras |
| VOCAL 5O. | Br. Pedro Ignacio Escalante pastor |
| SECRETARIO | Ing. Francisco Javier González López |

TRIBUNAL QUE PRACTICO EL EXAMEN
GENERAL PRIVADO

| | |
|---|---|
| DECANO | Ing. Julio Ismael González Podszueck |
| EXAMINADOR | Ing. Sergio Mario Silva Lorenzana |
| EXAMINADOR | Ing. Francisco Javier Guevara Castillo |
| EXAMINADOR | Ing. Orlando Sánchez Mendoza |
| SECRETARIO | Ing. Francisco Javier González López |

Guatemala, 9 de mayo de 1,996.

Ing. Calixto Monzón,
Coordinador de la carrera de Ingeniería
en Ciencias y Sistemas.

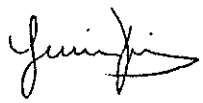
Ingeniero Monzón.

Por medio de la presente me permito hacer de su conocimiento que he procedido a revisar el trabajo de tesis titulado : **SISTEMAS DE BASES DE DATOS BASADOS EN LA LÓGICA**, elaborado por el estudiante **Bayron Wosvely López López**.

En mi calidad de asesor, he analizado el contenido, así como las conclusiones y recomendaciones expuestas y, a mi juicio, el mismo cumple con los objetivos propuestos para su desarrollo.

Sin otro particular, me suscribo de usted,

atentamente,



Ing. Francisco Javier Guevara Castillo

Guatemala, 30 de mayo de 1,996.

Ing. Calixto Monzón
Coordinador de la carrera de Ingeniería
en Ciencias y Sistemas

Estimado Ingeniero Monzón.

Por este medio me permito hacer de su conocimiento que he procedido a revisar el trabajo de tesis titulado : **SISTEMAS DE BASES DE DATOS BASADOS EN LA LÓGICA**, elaborado por el estudiante **Bayron Wosvely López López**, y a mi juicio el mismo cumple con los objetivos propuestos para su desarrollo.

Sin otro particular, me suscribo de usted,

atentamente,

Ing. Jorge Luis Alvarez

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Técnica y Regional de Post-grado de Ingeniería Sanitaria.

Ciudad Universitaria, zona 12
Guatemala, Centroamérica

Guatemala,
21 de junio de 1996

Ingeniero
Julio Ismael González Podszueck
Decano, Facultad de Ingeniería

Señor Decano:

Me dirijo a usted para informarle que después de conocer el dictamen del Asesor del trabajo de Tesis del estudiante BAYRON WOSVELY LOPEZ LOPEZ, titulado SISTEMAS DE BASES DE DATOS BASADOS EN LA LOGICA, procedo a la autorización del mismo.

Sin otro particular, me suscribo con las muestras de mi consideración y estima,

Atentamente.

UNIVERSIDAD DE SAN CARLOS
CIENCIAS DE LA COMPUTACION Y SISTEMA
FACULTAD DE INGENIERIA

Ing. Calixto Raúl Monzón P
Coordinador
Esc. de Ciencias y Sistemas

c.c. Archivo

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Técnica y Regional de Post-grado de Ingeniería Sanitaria.

Ciudad Universitaria, zona 12
Guatemala, Centroamérica

El decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la autorización por parte del coordinador de la Carrera de Ingeniería en Ciencias y Sistemas, al trabajo de tesis titulado: **SISTEMAS DE BASES DE DATOS BASADOS EN LA LÓGICA,** presentado por el estudiante BAYRON WOSVELY LÓPEZ LÓPEZ, procede a la autorización para la impresión de la misma.

IMPRIMASE

Ing. Julio Ismael González Podszueck
Decano

UNIVERSIDAD DE SAN CARLOS
DECANO
FACULTAD DE INGENIERIA
DE GUATEMALA

Guatemala, junio de 1996

**AGRADECIMIENTO**

De una forma especial al ingeniero Francisco Guevara, por su apoyo y colaboración para la elaboración de esta tesis.

A Noé Hernández , por su colaboración desinteresada.

**DEDICATORIA**

A Dios:
Porque el ha querido que llegue hasta aquí y al él le debo todo lo que soy.

A mis padres:
Thelma Yolanda López y Julio César López, como un pequeño agradecimiento por todos los sacrificios que han hecho por mí.

A mis amigos:
Ada, Claudia, Lydia, Rolando, Juan José, Luis y Víctor, ustedes son muy especiales para mí, los quiero.

# ÍNDICE GENERAL


| | Página |
|---|---|
| Lista de ilustraciones | iv |
| Glosario | v |
| Introducción | viii |
| CAPÍTULO UNO | |
| MODELOS DE DATOS | |
| 1.1 Modelo de datos tradicionales | 1 |
| 1.1.1 Eficiente acceso de archivos | 2 |
| 1.1.2 Lenguajes de búsquedas | 3 |
| 1.1.3 Manejo de transacciones | 4 |
| 1.1.4 Seguridad de los datos | 4 |
| 1.2 Terminología básica de un sistema de base de datos | 5 |
| 1.2.1 Niveles de abstracción de un DBMS | 5 |
| 1.2.2 El nivel físico | 5 |
| 1.2.3 El nivel conceptual | 5 |
| 1.2.4 El nivel de vista | 5 |
| 1.2.5 Esquemas e instancias | 7 |
| 1.2.6 Independencia de datos | 7 |
| 1.3 Modelo de entidad relación | 8 |
| 1.4 Modelo de datos relacional | 9 |
| 1.4.1 Llaves de las relaciones | 11 |
| 1.4.2 Relaciones con llaves comunes | 11 |
| 1.4.3 Operaciones en el modelo de datos relacional | 11 |
| 1.5 Álgebra relacional | 12 |
| 1.5.1 Operandos y operadores del Álgebra relacional | 12 |
| 1.5.2 Leyes del Álgebra | 14 |
| 1.6 Modelo de red | 14 |
| 1.7 Modelo Jerárquico | 15 |
| 1.8 Lenguajes de bases de datos | 16 |
| 1.8.1 Lenguajes de definición de datos | 16 |
| 1.8.2 Lenguajes de manipulación de datos | 17 |
| 1.8.3 Lenguajes anfitriones | 17 |
| 1.9 Arquitectura de un sistema de bases de datos | 18 |
| 1.10 Aplicaciones de sistemas de bases de datos | 19 |
| 1.11 Integración del lenguaje anfitrión y de manipulación de datos | 19 |
| 1.11.1 Lenguajes declarativos | 20 |
| 1.12 Sistemas orientados a objetos | 21 |
| 1.12.1 Identidad de objetos | 21 |
| 1.13 Historia y perspectiva | 22 |
| CAPÍTULO DOS | |
| LA LÓGICA COMO MODELO DE DATOS | |
| 2.1 Notación de la lógica | 24 |
| 2.1.1 Deducción | 25 |
| 2.1.2 Lenguajes de la lógica | 26 |
| 2.2 Reglas de inferencia | 27 |
| 2.3 Teorías | 28 |
| 2.3.1 Demostración de teoremas | 29 |
| 2.4 Demostración de teoremas | 30 |
| 2.5 Resolución | 32 |
| 2.6 Lógica de predicados | 35 |
| 2.7 Reglas de la lógica | 36 |
| 2.8 Expresividad de la lógica | 38 |

CAPÍTULO TRES
INTERPRETACIÓN DE LAS REGLAS DE LA LÓGICA
3.1 Prueba teórica de la interpretación de las reglas 39
3.2 Modelo teórico de la interpretación de las reglas 39
3.3 Definición computacional del significado 41
3.4 Comparación del significado 41

CAPÍTULO CUATRO
EL MODELO DE DATOS DATALOG
4.1 Estructura básica 42
4.2 Predicados intencionales y extensionales 42
4.3 Predicados construidos 43
4.4 Cláusulas y cláusulas de Horn 43
4.5 Grafo de dependencias y recursividad 44
4.6 Reglas seguras 46

CAPÍTULO CINCO
ESTRATEGIAS DE EVALUACIÓN DE REGLAS
5.1 Evaluación no recursiva de reglas 48
5.1.1 La relación definida por el cuerpo de una regla 48
5.1.2 Reglas restringidas 52
5.1.3 Computando las relaciones para predicados no recursivos 53
5.2 Computando las relaciones para predicados no recursivos 55
5.2.1 Puntos fijos de las ecuaciones de Datalog 56
5.2.2 Resolución de ecuaciones recursivas 58
5.3 Evaluación incremental de puntos fijos mínimos 60
5.4 Evaluación semi ascendente 62
5.5 Negaciones en el cuerpo de las reglas 64
5.5.1 No unicidad de puntos fijos 67
5.5.2 Negación estratificada 67
5.5.3 Búsqueda de estratificaciones 68
5.5.4 Reglas seguras estratificadas 71

CAPÍTULO SEIS
ÁLGEBRA RELACIONAL Y LÓGICA
6.1 De la lógica al Álgebra 75
6.2 Álgebra relacional monótona 77
6.3 Comparando Datalog y el Álgebra relacional 77
6.4 Cálculo relacional 78
6.4.1 Fórmulas del cálculo relacional 78
6.4.2 Dominio del cálculo relacional 80
6.4.3 Del Álgebra relacional al dominio del cálculo relacional 81
6.4.4 Fórmulas de dominio independiente 83
6.4.5 Fórmulas RCD seguras 83
6.4.6 Cálculo relacional de tupla 84
6.4.7 Tuplas seguras del cálculo relacional 86
6.5 Suposición de mundo cerrado 87

CAPÍTULO SIETE
LÓGICA CON FUNCIONES DE SÍMBOLOS
7.1 Evaluación de la lógica con funciones de símbolos 92
7.1.1 Igualando términos 95

7.2 Procesamiento descendente de la lógica 97
7.2.1 Razonamiento desde las metas 98
7.2.2 Seguimiento hacia atrás 98
7.3 Unificación 99
7.4 Arboles de reglas y metas 104
7.5 Proceso de las relaciones durante la expansión de los árboles de reglas y metas 108
7.5.1 Paso de información 109
7.5.2 Paso del encadenamiento de las reglas a las metas 109
7.5.3 Paso del encadenamiento de las metas a las reglas 109
7.5.4 Paso de las respuestas de las reglas a las metas 110
7.6 Grafos de reglas y metas 110
7.6.1 Adornos 110
7.6.2 Reglas adornadas 111
7.6.3 Construyendo un grafo de reglas y metas 111
7.7 Único patrón de encadenamiento reglas 114
7.8 Reorden delas submetas 116

CAPÍTULO OCHO
CONJUNTOS MÁGICOS
8.1 Técnica de reescritura de reglas 118
8.2 La única propiedad de encadenamiento 118
8.3 Predicados mágicos 119
8.4 Predicados suplementarios 119
8.5 Transformación básica de los conjuntos 119
8.6 Recursividad lineal 126

CAPÍTULO NUEVE
SISTEMAS DE BASES DE DATOS BASADOS EN LA LÓGICA
9.1 Aplicaciones de sistemas basados en la lógica 127
9.2 El sistema NAIL 130
9.2.1 NAIL, código fuente 131
9.2.2 Estrategia de selección 132
9.2.3 Captura de reglas 133
9.2.4 Base de datos estratégica 134
9.2.5 Código intermedio 135

Conclusiones 138
Recomendaciones 139
Bibliografía 140

# LISTA DE ILUSTRACIONES

Lista de figuras


| | | Página |
|---|---|---|
| Figura 1.1. | Niveles de abstracción de un sistema de base de datos. | 6 |
| Figura 1.2. | Diagrama de entidad relación. | 9 |
| Figura 1.3. | Una relación . | 10 |
| Figura 1.4. | Diagrama de un sistema de base de datos. | 20 |
| Figura 1.5. | Pasado y futuro de los sistemas de bases de datos. | 22 |
| Figura 4.1. | Ejemplo de un programa de lógica. | 44 |
| Figura 4.2. | Grafo de dependencias de la figura 4.1. | 45 |
| Figura 5.1. | Algoritmo de evaluación simple. | 59 |
| Figura 5.2. | La relación *P* para el predicado *padre*. | 59 |
| Figura 5.3. | Aplicación del algoritmo 5.3. | 60 |
| Figura 5.4. | Programa de evaluación semi-ascendente. | 63 |
| Figura 5.5. | Proceso de estratificación. | 69 |
| Figura 7.1. | Reglas para la definición de adición. | 90 |
| Figura 7.2. | Estado del mundo cerrado. | 91 |
| Figura 7.3. | Relación inicial para el predicado "verdadero". | 94 |
| Figura 7.4. | Algoritmo de igualación de términos. | 97 |
| Figura 7.5. | Prueba del árbol. | 98 |
| Figura 7.6. | Arboles de expresión para las fórmulas atómicas del ejemplo 7.8. | 102 |
| Figura 7.7. | Arboles de expresión para fórmulas no unificables | 103 |
| Figura 7.8. | La relación "P" para el predicado "par". | 105 |
| Figura 7.9. | Árbol de reglas y metas para la meta *arc(j,w)*. | 107 |
| Figura 7.10. | Grafo de reglas y metas para las reglas de la misma generación. | 113 |
| Figura 7.11. | Grafo mejorado de reglas y metas después de reordenar las submetas. | 115 |
| Figura 8.1. | Conjunto mágico, transformación de reglas. | 122 |
| Figura 8.2. | Grafo de reglas y metas para las reglas de una buena lista. | 123 |
| Figura 8.3. | Conjunto mágico, transformación de reglas para una buena lista. | 124 |
| Figura 8.4. | Grafo de reglas y metas para las reglas de caminos y longitudes. | 125 |
| Figura 8.5. | Conjuntos mágicos, transformación para las reglas de caminos. | 126 |
| Figura 9.1. | Camino para la definición de "D" para alcanzar el uso de "u". | 129 |
| Figura 9.2. | Reglas para la definición de alcance. | 129 |
| Figura 9.3. | Arquitectura de NAIL. | 131 |
| Figura 9.4. | Grafo de reglas y metas para el empleo de análisis de flujo de datos. | 133 |
| Figura 9.5. | Transformación lineal derecha sobre las reglas de ancestro | 136 |
| Figura 9.6. | ICODE para la evaluación de las reglas de ancestro. | 136 |


Lista de tablas


| | | |
|---|---|---|
| Tabla 2.1. | Símbolos de la lógica. | 27 |
| Tabla 2.2. | Cláusula C3:{P,S}. | 33 |

## GLOSARIO

AND : operador lógico "y".

AXIOMA DEDUCTIVO: regla de inferencia. Es una regla con la que dados ciertos hechos, se pueden deducir hechos adicionales.

BASE DE DATOS: sistema para archivar en computador. La base de datos en sí puede considerarse como una especie de archivo electrónico; dicho de otra manera, es un lugar donde se almacena un archivo de datos computarizados.

CAD : siglas en inglés que quieren decir Computer Aided Design , diseño auxiliado por computadora.

CASE : siglas en inglés que quieren decir Computer Aided Software Engineering, ingeniería de programas auxiliado por computadora.

CLÁUSULA: conjunto de literales que representan la disyunción de las mismas.

CLÁUSULA DE HORN: Cláusula que contiene como máximo una literal positiva.

CONJUNCIÓN : es el enunciado compuesto " $p$ y $q$", que se escribe "$P \wedge q$". Si $p$ y $q$ son enunciados.

CONOCIMIENTO : una colección integrada de hechos y relaciones que, cuando se ejercitan, producen un performance competente. La calidad y cantidad del conocimiento que posee una persona o computadora puede ser juzgado por la variedad de situaciones en que la persona o la computadora puede obtener resultados exitosos.

DATALOG : nombre asignado a un versión de Prolog aplicada a sistemas de bases de datos. La orientación es hacia la capacidad descriptiva, no hacia la capacidad computacional. Datalog destaca mucho más las consultas que las actualizaciones; aunque es posible y deseable, ampliar el lenguaje para manejar también las actualizaciones.

DBMS : siglas en inglés que quieren decir Database Management System, sistema de administración de base de datos.

DDL : siglas en inglés que quieren decir Data Definition Language, lenguaje de definición de datos.

DEDUCCIÓN : sinónimo de inferencia.

DISYUNCIÓN : es el enunciado compuesto " $p$ o $q$", que se escribe "$P \vee q$". Si $p$ o $q$ son enunciados.

DML : siglas en inglés que quieren decir Data manipulation Languaje, lenguaje de manipulación de datos.

EDB : siglas en inglés que quieren decir Extensional Database, base de datos extensional.

ENUNCIADO : sinónimo de proposición.



PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

FÓRMULAS : las fórmulas del cálculo proposicional se utilizan en los sistemas de bases de datos para representar la expresión condicional que define el resultado de una consulta.

HEURÍSTICA : un instrumento de simplificación que reduce o limita las búsquedas en un problema de espacio grande. A diferencia de los algoritmos, la heurística no garantiza soluciones correctas.

IDB : siglas en inglés que quieren decir Intensional Database, base de datos intencional.

INFERENCIA : el proceso mediante el cual nuevos hechos son derivados de los hechos conocidos . Una regla, combinada con una regla de inferencia y un hecho conocido resulta en un nuevo hecho.

KBMS : siglas en inglés que quieren decir Knowledge base management System, sistema de administración de bases de conocimientos.

LÓGICA : método formal de razonamiento. Como es formal, puede utilizarse para realizar tareas formales, como demostrar la validez de un argumento examinando sólo la estructura de ese argumento como una secuencia de pasos. En particular, por ser formal se puede mecanizar, es decir que se puede programar.

MODELO DE DATOS: una colección de tipos de objetos de información los cuales son las unidades básicas para construir cualquier base de datos que se ajuste al modelo. Una colección de reglas generales de integridad, las cuales limitan el conjunto de casos de esos tipos de objetos que puede aparecer en forma legal en cualquier base de datos que se ajuste al modelo. Una colección de operadores aplicables a esos casos de objetos para obtener información y para otros propósitos.

MODUS PONENS : nombre latino asignado específicamente a la regla de inferencia de separación.

NAIL : siglas en inglés que quieren decir Not Another Implementation of Logic, no otra implementación de lógica. NAIL es un sistema experimental basado en la lógica.

NOT : operador lógico "no".

TAUTOLOGÍA : enunciado que es verdadero para todos los posibles valores de sus variables proposicionales.

OBJETO : se refiere a entidades conceptuales o físicas que tienen varios atributos.

OO-DBMS: siglas en inglés que significan Object Oriented Data Base Management System, sistema de administración de base de datos orientado a objetos.

OR : operador lógico "o".

PREDICADO : es una función con un valor de verdad, es decir, una función que dados los argumentos apropiados, produce un valor falso o verdadero.

PROLOG: lenguaje de programación no procedural que se utiliza en aplicaciones de Inteligencia Artificial. Es un lenguaje simbólico de la Inteligencia Artificial basado en el cálculo de predicados. PROLOG es el lenguaje más

popular usado por investigadores de Inteligencia Artificial fuera de los estados unidos.

PROPOSICIÓN: una oración declarativa que es falsa o verdadera, pero no ambas a la vez.

REGLA : axiomas deductivos, que pueden ser formulados en un lenguaje.

SQL : siglas en inglés que quieren decir Structured Query Language, lenguaje de consulta estructurada.

TERMINO : es una proposición que: no incluye cuantificadores, o bien no incluye conectores lógicos o está encerrada entre paréntesis y se evalúa ya sea como falso o verdadero.

TRANSACCIÓN : es una unidad lógica de trabajo.

VLSI : siglas en inglés que quieren decir Very Large Scale Integration, integración a gran escala. El diseño VLSI se refiere al diseño y la construcción de circuitos electrónicos extremadamente densos y poderosos que pueden ser empacados en chips muy pequeños.

## INTRODUCCIÓN

Existe una tendencia en el campo de las bases de datos que se dirige hacia los sistemas de bases de datos basados en la lógica. La necesidad de explicar los términos, principios e ideas, se intenta satisfacer en este trabajo de tesis.

En Guatemala, los sistemas de bases de datos se pueden aplicar de varias formas; estos sistemas unidos a los sistemas de conocimiento pueden brindar una ayuda significativa, haciendo que se use la información de las bases de datos como conocimiento.

Sistemas de bases de datos basados en la lógica, es uno de los nombres con los que se encuentra en la literatura a estos sistemas, pero también se encuentra como base de datos lógica, sistema de administración de base de conocimientos, DBMS experto, etc. Estos sistemas deben ser usados en aplicaciones donde los datos existen de una manera masiva y donde se hace necesario tener una facilidad de búsqueda más poderosa que las que ofrecen los lenguajes de búsqueda, típicos, como SQL.

La estructura de este trabajo es la siguiente: el capítulo uno, presenta los modelos tradicionales de bases de datos, mostrando un panorama general del tema; este capítulo es necesario para mostrar los conceptos y términos habituales de las bases de datos. En el capítulo dos, se muestra la lógica como modelo de datos, el objetivo, aquí, es explicar las nuevas ideas de la lógica en términos convencionales de las bases de datos.

Los capítulos tres, cuatro y cinco, muestran de una forma detallada un marco teórico de los conceptos básicos para entender los sistemas basados en la lógica. Para desarrollar esta parte se tomó como base el modelo de datos de Datalog, nombre asignado para relacionar una versión de Prolog adaptado para sistemas de bases de datos.

En el capítulo seis, se muestran las ventajas y desventajas del uso de funciones de símbolos en las reglas, un estilo distinto al de Datalog, donde se puede presentar la recursividad en las reglas de la lógica. Además, presenta un estudio preparatorio para la notación especializada que se usa en los capítulos siete y ocho.

En el capítulo siete, se explica la técnica de los conjuntos mágicos para la reescritura de las reglas de la lógica, donde se ofrecen nuevas ventajas asociadas a las técnicas de evaluación.

Finalmente, el capítulo ocho pretende mostrar las áreas de aplicación de los sistemas de bases de datos basados en la lógica y explica, con poco detalle, un proyecto de este tipo que se encuentra en desarrollo.

# 1. MODELO DE DATOS

Existen dos cualidades que distinguen a los sistemas de bases de datos, de otros tipos de sistemas de programación.

1. La habilidad de manejar datos persistentes y
2. La habilidad de accesar largas cantidades de datos, eficientemente.

El punto (1) se refiere solamente a que hay una base de datos y que existe, permanentemente. El punto (2) distingue a un DBMS (Data Base Management System) de un sistema de archivos, que también maneja datos persistentes, pero, que generalmente, no ayuda a proveer rápido acceso a porciones arbitrarias de datos. Las capacidades de un DBMS son necesarias mayormente cuando la cantidad de datos es muy grande, porque para cantidades pequeñas de datos, técnicas simples de acceso, tales como búsquedas lineales de datos, son usualmente adecuadas.

Las dos propiedades de un DBMS, mencionadas anteriormente, pueden ser consideradas como fundamentales, pero, existen otras capacidades que deben ser encontradas casi universalmente en todos los DBMS comerciales. Estas son:

a) soportar, por lo menos un modelo de datos o abstracción matemática a través de la cual el usuario puede visualizar los datos;

b) soportar para ciertos lenguajes de alto nivel que permiten al usuario la definición de estructuras de datos, acceso de datos y la manipulación de datos;

c) manejo de transacciones, la capacidad de proveer un correcto y común acceso a la base de datos por muchos usuarios al mismo tiempo;

d) control de acceso, la habilidad de limitar el acceso a los datos por usuarios sin autorización y la habilidad de verificar la validez de los datos;

e) elasticidad, la habilidad de recuperarse de caídas del sistema sin pérdida de datos.

## 1.1 Modelo de datos tradicionales.

Cada DBMS provee, por lo menos, un modelo abstracto de los datos, que permiten al usuario visualizar la información no como una colección de bits, sino en términos más comprensibles. Los modelos de datos especifican como piezas de información separadas están relacionadas en una base de datos. Es posible visualizar los datos en varios niveles de abstracción. En un nivel relativamente bajo, un DBMS comúnmente permite visualizar los datos, como un conjunto de archivos.

Un modelo de datos es un formalismo matemático con dos partes:

1. una notación para la descripción de datos y

2. un conjunto de operaciones usadas para manipular esos datos.

Se consideran aquí los principales modelos usados en los sistemas de bases de datos; el modelo de entidad relación, el modelo relacional, el modelo de red y el modelo jerárquico.

**Ejemplo 1.1:** una corporación normalmente, debería, conservar un archivo concerniente a sus empleados y los registros de cada empleado deberían tener los campos para su nombre, apellido, número de identificación del empleado, salario, dirección y probablemente otra docena de piezas de información . Para este simple ejemplo, supóngase que se tiene en el registro sólo el nombre del empleado y el nombre del jefe del empleado. La estructura del registro debería parecerse a:

```
registro
      nombre: caracter[30];
      jefe    : caracter[30];
end
```

El archivo en sí es una secuencia de registros, uno para cada empleado de la compañía. Algunas veces, lo que se llama de una forma abstracta: relación, es un archivo de registros, que debe ser descrita como:

EMPLEADOS(NOMBRE,JEFE)

donde EMPLEADOS es el nombre de la relación, correspondiente al archivo mencionado en el ejemplo 1.1 NOMBRE y JEFE son nombres de campos, estos campos se llaman frecuentemente *atributos*, cuando estamos hablando de relaciones.

Pareciera que los términos, archivo y relación, son sinónimos, pero se debe estar alerta ya que son dos conceptos distintos. Una relación es una abstracción de un archivo donde el tipo de los campos se conoce superficialmente y el orden entre los registros no está especificado. Los registros en una relación son llamados *tuplas*. Así, un archivo es un conjunto de registros y una relación es un conjunto de tuplas.

### 1.1.1 Eficiente acceso de archivos.

La habilidad de almacenar un archivo, no es remarcable; el sistema de archivos asociado con el sistema operativo hace eso. La capacidad de un DBMS se observa cuando se hace el acceso de los datos de un archivo. Supóngase que se quiere encontrar al jefe del empleado "Juan Pérez". Si la compañía tiene miles de empleados, resulta caro recorrer el archivo entero para encontrar un registro con NOMBRE = "Juan Pérez". Un DBMS ayuda a tener archivos indexados o índices, que permiten accesar el registro para "Juan Pérez" directamente no importando qué tan grande es el archivo. De la misma manera, la inserción de un nuevo registro o la eliminación de existentes está acompañado de un tiempo pequeño y constante independiente del tamaño del archivo.

Otra ventaja que proporciona un DBMS es la capacidad de navegar entre archivos, esto es, combinar valores en dos o más archivos para obtener la información que se desea.

**Ejemplo 1.2 :** supóngase que se almacenó en el registro de un empleado, el departamento para el que trabaja, pero no su jefe. En otro archivo, llamado DEPARTAMENTOS, se tienen registros que asocian los nombres de los departamentos con el jefe, en la forma de relaciones se tiene:

EMPLEADOS(NOMBRE,DEPTO)
DEPARTAMENTOS(DEPTO,JEFE)

Ahora si se quiere encontrar al jefe de "Juan Pérez", se necesita navegar desde EMPLEADOS a DEPARTAMENTOS, usando la igualdad del campo DEPTO en

ambos archivos. Primero se encuentra el registro en el archivo de EMPLEADOS que tiene el NOMBRE= "Juan Pérez" y de ese registro se obtiene el valor de DEPTO, y, si se asume que es "publicidad", entonces, se busca en el archivo de DEPARTAMENTOS el registro que tiene DEPTO = "publicidad" y, allí, se encuentra que JEFE ="Pedro Matute". Si se tienen bien definidos los índices, se puede ejecutar cada uno de estos accesos en un tiempo corto y constante, independientemente de la longitud de los archivos.

### 1.1.2 Lenguajes de búsqueda.

Para hacer más fácil el acceso de los archivos, un DBMS provee un lenguaje de búsqueda o un lenguaje de manipulación de datos, para expresar operaciones sobre los archivos. Los lenguajes de búsqueda se distinguen en el nivel de detalle que ellos requieren del usuario, con sistemas, basados en el modelo de datos relacional, generalmente, requieren menos detalle que los lenguajes basados en otros modelos.

**Ejemplo 1.3:** la búsqueda discutida en el ejemplo 1.2, "encontrar al jefe de Juan Pérez", puede ser escrita en el lenguaje SQL, que está basado en el modelo de datos relacional, de la siguiente forma:

```
(1) SELECT JEFE
(2) FROM EMPLEADOS, DEPARTAMENTOS
(3) WHERE EMPLEADOS.NOMBRE = 'Juan Pérez'
(4)         AND EMPLEADOS.DEPTO = DEPARTAMENTOS.DEPTO;
```

La línea (1) le indica al DBMS que imprima el nombre del jefe como una respuesta, la línea (2) indica que busque en las relaciones de EMPLEADOS y DEPARTAMENTOS, (3) indica que el nombre del empleado es 'Juan Pérez' y la última línea indica que el Jefe está conectado al empleado por la asociación (en la relación de DEPARTAMENTOS) con el mismo departamento con el que el empleado está asociado (en la relación de EMPLEADOS).

A continuación, se observa la misma búsqueda escrita en una versión simplificada del lenguaje de búsqueda del modelo de red:

```
(1) EMPLEADOS.NOMBRE := "Juan Pérez"
(2) FIND EMPLEADOS RECORD BY CALC-KEY
(3) FIND OWNER OF CURRENT EMP-DEP SET
(4) FIND FIRST JEFE RECORD IN CURRENT  DEPT-JEF SET
(5) print
```

Las líneas (1) y (2) juntos informan al DBMS que encuentre el registro para Juan Pérez en el archivo de EMPLEADOS. La línea (3) utiliza una estructura EMP-DEPT que conecta los empleados con sus departamentos, para encontrar el departamento al que pertenece el empleado. La línea (4) indica que existe otra estructura DEPT_JEF, relacionando los departamentos con los jefes, en la línea (5) se encuentra una impresión del primer jefe listado para el departamento de Juan Pérez y, técnicamente, se debería buscar jefes adicionales para el mismo departamento. Hay que hacer notar que la operación de impresión de la línea (5) no es parte del lenguaje de búsquedas, pero, parte del lenguaje anfitrión circundante, que es un lenguaje ordinario de programación.

### 1.1.3 Manejo de transacciones.

Otra capacidad importante de un DBMS es la habilidad de manejar simultáneamente grandes cantidades de transacciones, que son procedimientos operando sobre la base de datos. Algunas bases de datos son tan grandes, que sólo pueden ser usadas simultáneamente por varias computadoras; algunas de estas computadoras pueden estar dispersas alrededor de un país o el mundo. Por ejemplo, el sistema de bases de datos usado por un banco, accesada casi instantáneamente por cientos o miles de cajeros automáticos, también como un número igual o mayor de empleados en las sucursales o agencias bancarias. Un sistema de reservaciones de una línea aérea es otro buen ejemplo.

Algunas veces, dos accesos no interfieren uno con el otro. Por ejemplo, cualquier número de transacciones puede estar leyendo el balance del banco sin tener inconsistencias. Pero, si al mismo tiempo de hacer un deposito, se está extrayendo dinero de otra sucursal en la misma cuenta, el resultado de las dos transacciones ocurridas simultáneamente y sin coordinación, es impredecible. Las transacciones que están modificando los datos, deben desechar otras transacciones tratando de leer o escribir ese registro al mismo tiempo. Un DBMS debe, por lo tanto, proveer alguna forma de control de concurrencia para prevenir el acceso no coordinado a los mismos datos por más de una transacción.

Problemas más complejos ocurren cuando la base de datos está distribuida en diferentes sistemas de computación, aunque con duplicación de datos para permitir fácil acceso local de los datos y proteger los datos contra la destrucción de datos, si alguna computadora se cae.

### 1.1.4 Seguridad de los datos.

Un DBMS no sólo debe proteger la información contra pérdidas de datos cuando el sistema se cae, como se mencionó anteriormente, sino, debe prevenir accesos no autorizados. Por ejemplo, sólo usuarios con cierta autorización deberían tener acceso al campo de salario de un archivo de empleados; y, el DBMS debe ser capaz de asociar a los usuarios, sus privilegios para ver archivos, campos con archivos y otras características de la base de datos. Así, un DBMS debe mantener una tabla para saber qué privilegios tiene un usuario para cada objeto. Por ejemplo, un usuario puede tener el privilegio de leer un archivo pero no de insertar o eliminar datos; otro podría tener acceso a leer y modificar el archivo.

Para proveer un conjunto adecuado de construcciones, de tal manera que los usuarios puedan ver parte de los archivos sin necesidad de mostrar todo el archivo, un DBMS algunas veces provee la facilidad de *vistas*, que permite crear objetos imaginarios definidos de una forma precisa de los objetos reales.

**Ejemplo 1.4:** supóngase que se tiene un archivo de EMPLEADOS con los siguientes campos:

```
EMPLEADOS(NOMBRE,DEPARTAMENTO,SALARIO,DIRECCION)
```

y se pretende que la mayoría de las personas tengan acceso a los campos menos al de SALARIO. En el lenguaje de SQL, se puede definir una vista así:

```
CREATE VIEW SEGURIDAD BY
SELECT NOMBRE,DEPARTAMENTO,DIRECCION
FROM EMPLEADOS;
```

esta vista SEGURIDAD está integrada de los campos NOMBRE, DEPARTAMENTO y DIRECCIÓN del archivo de EMPLEADOS. SEGURIDAD debe ser imaginada como una relación descrita por

```
SEGURIDAD(NOMBRE,DEPARTAMENTO,DIRECCION)
```

la vista SEGURIDAD no existe físicamente como un archivo, pero puede ser usada para hacer búsquedas como si realmente existiera. Por ejemplo, se puede buscar el departamento de Juan Pérez, en SQL:

```
SELECT DEPARTAMENTO
FROM SEGURIDAD
WHERE NAME='Juan Pérez'.
```

Los usuarios normales están permitidos para accesar la vista SEGURIDAD, pero, no la relación de EMPLEADOS. Los usuarios que pueden tener el privilegio de consultar o conocer los salarios, tienen un privilegio distinto de los que pueden cambiar los salarios.

## 1.2 Terminología básica de un sistema de base de datos.

### 1.2.1 Niveles de abstracción en un DBMS.

Entre la computadora, relacionada con los bits, y los usuarios finales relacionados con los conceptos de empleados, cuentas bancarias o asientos de una línea aérea, existirán varios niveles de abstracción. Desde un punto de vista estándar considerando los niveles de abstracción se muestra en la figura 1.1. En el mundo de los sistemas de bases de datos, generalmente no hay una razón para definir el nivel de bit o byte.

### 1.2.2 El nivel físico.

Una colección de archivos y los índices o, bien, otras estructuras de almacenamiento usadas para accesarlos eficientemente es llamado, *base de datos física* . La base de datos física reside, permanentemente, en dispositivos de almacenamiento secundarios, tal como discos. Varias bases de datos físicas diferentes pueden ser manejadas por el mismo software del DBMS.

### 1.2.3 El nivel conceptual.

La base de datos conceptual es una abstracción del mundo real como si perteneciera a los usuarios de la base de datos. Un DBMS provee un lenguaje de definición de datos o DDL, para describir el esquema conceptual y la implementación del esquema conceptual por el esquema físico. El lenguaje de definición de datos permite definir la base de datos conceptual en términos de un modelo de datos. Por ejemplo, el modelo relacional. En ese modelo los datos parecen tablas, cuyas columnas están encabezadas por atributos y sus filas son *tuplas* que son similares a los registros.

### 1.2.4 El nivel de vista.

Una vista o subesquema es una porción de la base de datos conceptual o, bien, una parte abstracta de la base de datos conceptual. La mayoría de los sistemas manejadores de bases de datos proveen una facilidad para manejar vistas llamado *lenguaje de definición de subesquema de datos* y una facilidad para expresar búsquedas y operaciones sobre las vistas, que podría llamarse *lenguaje de manipulación de subesquema de datos*. En esencia, la construcción de vistas es el proceso inverso de integración de la base de datos, para cada colección de datos que contribuye a la base de datos conceptual se debe construir una vista conteniendo sólo dichos datos. Las vistas también son importantes para reforzar la seguridad en un sistema de base de datos, permitiendo, así, que los usuarios puedan observar sólo un subconjunto de los datos. El ejemplo 1.4 ilustra el uso de estas vistas.
Representan asociaciones entre ambos archivos.

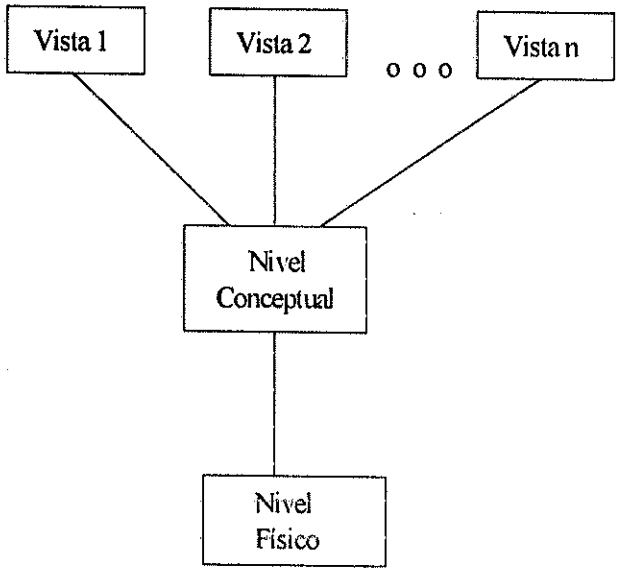



**FIGURA 1.1 Niveles de abstracción de un sistema de base de datos.**

Algunas veces una vista es sólo una base de datos conceptual pequeña y se encuentra al mismo nivel de abstracción que la base de datos. Sin embargo, tiene sentido que las vistas pueden ser más abstractas que una base de datos conceptual, como cuando los datos relacionados con una vista pueden ser construidos de la base de datos conceptual porque no se encuentran presentes en la base de datos actual.

**Ejemplo 1.6:** para enfatizar la diferencia entre los niveles de abstracción físicos, conceptual y de vistas , se usará una analogía del mundo de los lenguajes de programación. En particular, de los arreglos; en el nivel conceptual se debería describir un arreglo por una declaración como:

*integer array A[1..n,1..m]*

mientras que el nivel físico para ver este arreglo debe ser como un bloque consecutivo de almacenamiento y con la regla:

*A[i,j] está localizada en* $a_0+4(m(i-1)+j-1)$

Una vista del arreglo *A* podría estar formado por la declaración de la función *f(i)* para hacer la suma de *j = 1 a m* de *A[i,j]*. En esta vista, no solamente se ve a *A* relacionada de una forma distinta, ya que como función y no como arreglo se ha obscurecido un poco de información, ya que sólo se pueden ver las sumas de las filas en lugar de ver las mismas filas.

### 1.2.5 Esquemas e instancias.

En adición a los grados en los niveles de abstracción implicados en la figura 1.1, existe otra, una dimensión ortogonal de la percepción de las bases de datos. Cuando la base de datos es diseñada el interés son los planes para la base de datos, cuando ya está en uso, lo importante son los datos presentes en la base de datos. Hay que notar que los datos en la base de datos cambian frecuentemente.

El contenido actual de una base de datos se puede determinar como una instancia de la base de datos. El término extensión de la base de datos y estado de la base de datos pueden ser usados como sinónimos. Sin embargo, el término base de datos extensional se usa cuando se está hablando de los sistemas basados en la lógica.

Los planes para una base de datos indica los tipos de entidades con los que tendrá relación, las relaciones entre estos tipos de entidades y la forma que las entidades y relaciones a un nivel de abstracción están expresadas en el nivel inmediato inferior. El término esquema es usado para referirse a los planes, así, cuando se habla del esquema conceptual es como hablar de los planes para la base de datos conceptual y se dice, los planes de la base de datos física para indicar el esquema físico. Los planes para una vista algunas veces se refieren simplemente como un subesquema. El término *intención* es algunas veces usado para indicar *esquema*.

**Ejemplo 1.7**: para continuar con la analogía de los arreglos del ejemplo 1.6, la descripción de los arreglos y funciones dados en ese ejemplo son en realidad un esquema de información.

1. El esquema físico es la sentencia:

   $a_0+4(m(i-1)+j-1)$

   que el arreglo *A* está almacenado iniciando en la localidad $a_0$ y que *A[i,j]* aparece en esa palabra

2. El esquema conceptual es la declaración:

   *integer array A[1..n,1..m]*

   *A* es un arreglo de entero con *n* filas y *m* columnas.

3. El subesquema es la definición de la función *f*, que es

   *f(i) = sumatoria de j=2 hasta m de A[i,j]*

### 1.2.6 Independencia de datos.

PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

La cadena de abstracciones que muestra la figura 1.1, de base de datos de vista a conceptual a física, provee dos niveles de independencia de datos. Obviamente, en el diseño de una base de datos el esquema físico puede ser cambiado sin necesidad de alterar el esquema conceptual o la necesidad de la redefinición de subesquemas. Esta independencia se refiere a la independencia de datos física. Esto implica que las modificaciones a la organización física de la base de datos puede afectar la eficiencia de los programas de aplicación, pero no será necesario reescribir esos programas solamente porque la implementación del esquema conceptual ha sido cambiado por el esquema físico. El valor de la independencia de base de datos física es que permite que se ajuste la base de datos física para la eficiencia mientras los programas de aplicación pueden correr como si no hubiera ocurrido algún cambio.

Las relaciones entre las vistas y la base de datos conceptual también proveen un tipo de independencia llamada *independencia de datos lógica*, conforme la base de datos es usada, es necesario cambiar el esquema conceptual, por ejemplo, al agregar información acerca de diferentes tipos de entidades o información extra de entidades ya existente; algunas modificaciones al esquema conceptual pueden hacerse sin necesidad de modificar los subesquemas existentes y otras modificaciones al esquema conceptual pueden hacerse si se redefine el mapeo de los subesquemas al esquema conceptual. Nuevamente el cambio de los programas de aplicación no es necesario. El único tipo de cambios en el esquema conceptual, que puede no reflejarse en la redefinición de un subesquema en términos del esquema conceptual, es la eliminación de información que corresponde a la información presente en el subesquema. Tales cambios, naturalmente, requerirán que se reescriban los programas de aplicación o que algunos se desechen.

## 1.3 Modelo de entidad relación.

El propósito del modelo de entidad relación es permitir la descripción del esquema conceptual de una empresa, sin tomar en cuenta la eficiencia o el diseño físico de la base de datos. Puede asumirse normalmente que el diagrama de entidad relación construido, se convierta más tarde en un esquema conceptual de uno de los otros modelos.

### Entidades

El término entidad define una formal definición, una entidad es un objeto o cosa que existe y que puede distinguirse de otras entidades. Una entidad puede ser una persona una institución, un vuelo, etc.

### Atributos

Es necesario caracterizar a todos los miembros de una entidad a través de una colección de características llamadas atributos o propiedades. Estas propiedades se asocian con cada entidad en un conjunto de valores de un dominio de valores. Usualmente, el dominio para un atributo será un conjunto de enteros, números reales o una lista de caracteres.

En términos de una construcción regular de una base de datos, las entidades generalmente están formadas por registros mientras que sus atributos están representados como los campos de esos registros.

### Llaves

Un atributo o un conjunto de atributos, cuyos valores identifican de forma única una tupla, es llamada llave para esa entidad. En principio, se dice que cada entidad tiene una llave desde que se asume que cada entidad es distinguible de las otras.

**Relaciones**

Una relación entre entidades es una lista ordenada de entidades. Una entidad particular puede aparecer más de una vez en una lista. Esta lista de entidades es la noción del nivel de esquema de una relación.

Si existe una relación $R$ entre las entidades $E_1, E_2, \ldots, E_k$, entonces, la instancia actual de $R$ es un conjunto de $k$-tuplas, a la que se llamará una relación. Cada $k$-tupla $(e_1, e_2, \ldots, e_k)$, en la relación $R$ implica esas entidades $e_1, e_2, \ldots, e_k$, donde $e_1$ está en el conjunto $E_1$, $e_2$ está en el conjunto $E_2$ y así sucesivamente, están en la relación $R$ como un grupo.

Es usual resumir la información en un diseño usando diagramas de entidad relación, donde:

1. los rectángulos representan entidades;

2. los atributos están asociados a los rectángulos. Se encuentran unidos a sus entidades por medio de aristas. Algunas veces los atributos que forman parte de la llave de la entidad se muestran con asterisco;

3. las relaciones tienen un nombre y se encuentran unidas a las entidades que las constituyen por medio de aristas.

En la figura 1.2 se muestra un diagrama simple de entidad relación, con tres entidades, EMPLEADO, DEPARTAMENTO y JEFE. Los primeros dos están relacionados por la relación ASIGNADO_A y el segundo y tercero están relacionados por ADMINISTRA. Se muestran cuatro atributos, ID, NOMBRE, TELEFONO y SALARIO, para EMPLEADO; ID se toma como llave. DEPARTAMENTO tiene los atributos COD, NOMBRE y LOCALIZACIÓN, mientras que JEFE tiene solamente el ID y el NOMBRE.

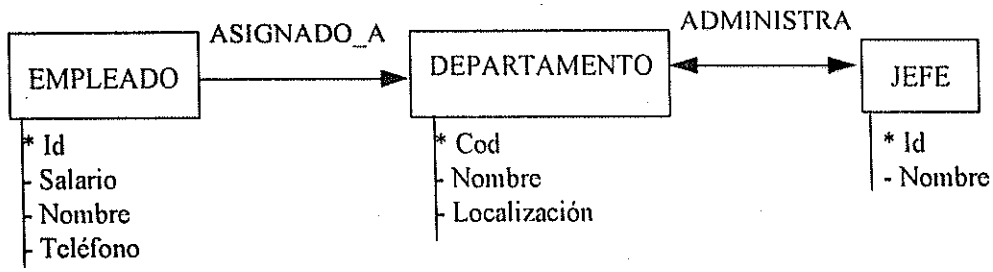



**Figura 1.2 Diagrama de entidad relación.**

**1.4 Modelo de datos relacional.**

El modelo relacional, aunque no es el modelo de datos usado en el primer sistema manejador de bases de datos ha crecido lentamente en importancia desde su exposición por E. Codd en 1970, al punto donde es el modelo de preferencia para la implementación de nuevas bases de datos. Tal vez la más importante razón para la popularidad del modelo es su poder, su lenguaje simple y declarativo con el que las operaciones de los datos son expresadas.

El concepto matemático fundamental del modelo relacional es el conjunto de relaciones teóricas, que son un subconjunto del producto cartesiano de la lista de dominios. Formalmente, un dominio es un conjunto de valores, no un tipo de datos. El producto cartesiano, o sólo producto, de dominios $D_1, D_2, \ldots, D_k$, escrito $D_1 X D_2 X \ldots X D_k$ es el conjunto de todas las $k$-tuplas $(v_1, v_2, \ldots, v_k)$ tal que $v_1$ está en $D_1$, $v_2$ está en $D_2$, y así, sucesivamente. Por ejemplo, si se tiene $k = 2$, $D_1 = \{0,1\}$, y $D_2 = \{a,b,c\}$, entonces $D_1 X D_2$ es $\{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$.

Una relación es cualquier subconjunto del producto cartesiano para uno o más dominios. Se debe asumir que una relación es finita a menos que ocurra lo contrario. Por ejemplo, $\{(0,a),(0,c),(1,b)\}$ es una relación, un subconjunto del producto $D_1 X D_2$ mencionado anteriormente. El conjunto vacío es otro ejemplo de una relación.

Los miembros de una relación son llamados tuplas. Cada relación que es un subconjunto de algún producto $D_1 X D_2 X \ldots X D_k$ de $k$ dominio se dice que tiene grado, $k$. Una tupla $(v_1, v_2, \ldots, v_k)$ tiene $k$ componentes. Una tupla con $k$ componentes generalmente, es llamada una $k$-tupla. Esto ayuda a visualizar una relación como una tabla, donde cada fila es una tupla y cada columna corresponde a un componente. A las columnas generalmente se les asignan nombres llamados atributos. El conjunto de los nombres de los atributos, para una relación es llamada el esquema de la relación. Si nosotros llamamos a una relación REL y el esquema de esta relación, tiene los atributos $A_1, A_2, \ldots, A_k$, entonces, se escribe el esquema de la relación como $REL(A_1, A_2, \ldots, A_k)$.

Ejemplo 1.2: en la figura 1.3 se ve una relación cuyos atributos son DEPARTAMENTO, MUNICIPIO y POBLACIÓN. El grado de esta relación es tres. Por ejemplo,

(Guatemala,Guatemala, 12.879)

es una tupla. El esquema de la relación para esta relación es (DEPARTAMENTO,MUNICIPIO,POBLACION); si la relación fuera llamada INFORME, debería escribirse el esquema de la relación como INFORME(DEPARTAMENTO,MUNICIPIO,POBLACION).

| DEPARTAMENTO | MUNICIPIO | POBLACIÓN |
|---|---|---|
| Guatemala | Guatemala | 125,839 |
| Quetzaltenango | Xela | 12,125 |
| Peten | Santa Elena | 9,583 |

**Figura 1.3 Una relación**

En otras palabras, puede decirse que una base de datos relacional es aquella cuyos usuarios la perciben como un conjunto de tablas y nada más que tablas, donde todo el contenido de la información de la base de datos se representa como valores explícitos de los datos; este método de representación es el único en una base de datos relacional.

### 1.4.1 Llaves de las relaciones.

De la misma forma que en los grupos de entidades, las relaciones tienen conjuntos de uno o más atributos que sirven como llaves. Para las relaciones puede darse una definición de llave que es más formal que la noción de un conjunto de atributos que distinguen los miembros de una entidad. Se dice que un conjunto *S* de atributos de una relación *R* es una llave si

1. no hay una instancia de *R* que represente un estado posible donde se puedan tener dos tuplas que concuerdan en todos los atributos de *S*, y, no son la misma tupla; y,

2. ningún subconjunto propio de *S* tiene la propiedad (1).

Es importante recordar que las llaves dependen del esquema, no de los valores de la relación. También se debe observar que una relación puede tener más de una llave . Generalmente, cuando una relación tiene dos o más llaves se toma una y se reconoce esa como la única llave. Por consiguiente, el término de llave primaria se utilizará para referirse a la llave seleccionada entre las otras opciones, las cuales serán llamadas llaves candidatas.

### 1.4.2 Relaciones con llaves comunes.

Cuando dos relaciones tienen una llave candidata en común, se pueden combinar los atributos de las dos esquemas de relación y reemplazar las dos relaciones por una, cuyo conjunto de atributos es la unión de dos conjuntos. Una ventaja de hacer esto es que se ahorra espacio de almacenamiento necesario para repetir los valores de las llaves en dos relaciones. Además, es que las consultas acerca de los atributos de las dos relaciones pueden ser contestadas algunas veces más rápidamente si las dos relaciones se combinan.

### 1.4.3 Operaciones en el modelo de datos relacional.

Aquí se introduce la familia de operaciones, usualmente, asociadas con este modelo, existen dos tipos diferentes de notaciones usadas para expresar las operaciones en las relaciones:

1 la notación algebraica, llamada Álgebra relacional, donde las consultas son expresadas por la aplicación especializada de operadores a las relaciones y
2. la notación lógica, llamada Cálculo relacional, donde las búsquedas son expresadas por la escritura de fórmulas lógicas que las tuplas en las respuestas deben satisfacer.

El Álgebra relacional incluye algunas operaciones familiares como la unión y diferencia de relaciones, pero, también, incluye algunas que talvez no son familiares. La notación lógica, se trata con mayor detalle en el siguiente capítulo. Uno de los factores interesantes acerca de estas notaciones para las bases de datos relacionales es que son equivalentes en su

poder expresivo, es decir, que cada uno puede expresar cualquier consulta que el otro puede , pero no más.

## 1.5 Álgebra relacional.

### 1.5.1 Operandos y operadores del Álgebra relacional.

Las operaciones del Álgebra relacional pueden ser relaciones constantes o variables denotando relaciones de un rango fijo. Los grados asociados a una variable serán mencionadas solamente cuando ésta sea importante. Existen cinco operaciones que sirven para definir el Álgebra relacional.

1 Unión: la unión de relaciones $R$ y $S$, denotado como $R \cup S$, es el conjunto de tuplas que están en $R$ o $S$ o en ambos. El grado de $R$ y $S$ debe ser el mismo. Los nombres de los atributos de las relaciones operándose son ignorados cuando se hace la unión y la relación resultante puede estar dada por atributos arbitrarios. El orden de los atributos en los operandos se considera cuando se hace la unión; y, lo mismo ocurre con todos los otros operadores.

2. Diferencia: la diferencia de relaciones $R$ y $S$, denotado como $R-S$, es el conjunto de tuplas en $R$ que no están en $S$. Nuevamente, se requiere que $R$ y $S$ tengan el mismo grado.

3. Producto cartesiano: sea $R$ y $S$ relaciones de grado $k_1$ y $k_2$, respectivamente. Entonces, $R \times S$, el producto cartesiano de $R$ y $S$, es el conjunto de todas la posibles tuplas$(k_1 + k_2)$ donde sus primeros $k_1$ componentes forman una tupla en $R$ y sus últimos $k_2$ componentes forman una tupla en $S$.

4. Proyección: la idea detrás de esta operación es que se toma una relación $R$, se remueven algunos de los componentes (atributos) y/o se arreglan algunos de los componentes restantes. Si $R$ es una relación de grado $k$, entonces $\pi i_1, i_2, \ldots i_m(R)$, donde las $i_j$ son distintos enteros en el rango de 1 a $k$, denota la proyección en los componentes $i_1, i_2, \ldots i_m$, que es el conjunto de tuplas$(m)$ $a_1, a_2, \ldots a_m$ de tal manera que existe alguna tupla$(k)$ $b_1, b_2, \ldots b_k$ en $R$ donde $a_j = b_{ij}$ para $j=1,2,\ldots,m$. Por ejemplo, $\pi_{3,1}(R)$ es procesado tomando cada tupla $\mu$ en $R$ y formando una tupla de 2 columnas del tercer y primer componente de $\mu$ en ese orden. Si $R$ tiene atributos etiquetando sus columnas, entonces, se pueden substituir los nombres de los atributos por los números de los componentes y se puede usar el mismo nombre del atributo en la relación proyectada. Así, si la relación $R$ es $R(A,B,C,D,)$, entonces $\mu_{3,1}(R)$ y la relación resultante tiene su atributo $C$ nombrando su primera columna y el atributo $A$ nombrando su segunda columna.

5. Selección: sea $F$ una fórmula que incluye:
   i) operandos que son constantes o números de componentes, el componente $i$ es representado por $\$i$,

   ii) los operadores aritméticos de comparación <,=,>,<=,=|, y >=

   iii) los operadores lógicos (y),(o) y (no).

entonces $\sigma_F(R)$ es el conjunto de tuplas $\mu$ en $R$ tal que para todo $i$, se substituye el *i-esimo* componente de $\mu$ para cada ocurrencia de $i en la fórmula $F$, la fórmula $F$ se convierte en verdadera.

**Ejemplo 1.9**: sean $R$ y $S$ las dos relaciones:

| A | B | C |
|---|---|---|
| a | b | c |
| d | a | f |
| c | b | d |

**(a) Relación *R***

| D | E | F |
|---|---|---|
| b | g | a |
| d | a | f |

**(b) Relación *S***

En las siguientes figuras se pueden ver las relaciones $R \cup S$ y $R - S$, debe notarse que se pueden hacer uniones y diferencias aunque las columnas de las dos relaciones tienen diferentes nombres, pero, sí el mismo número de componentes.

| | | |
|---|---|---|
| a | b | c |
| d | a | f |
| c | b | d |
| b | g | a |

**(a) $R \cup S$**

| | | |
|---|---|---|
| a | b | c |
| c | b | d |

**(b) $R - S$**

La siguiente figura muestra la operación $R \times S$. Como $R$ y $S$ no tienen ninguna conexión entre sus atributos, se pueden tomar los nombres de las columnas para esta operación.

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| a | b | c | b | g | a |
| a | b | c | d | a | f |
| d | a | f | b | g | a |
| d | a | f | d | a | f |
| c | b | d | b | g | a |
| c | b | d | d | a | f |

***R X S***

Las siguientes figuras muestran otras dos operaciones del Álgebra relacional:

| A | C |
|---|---|
| a | c |
| d | f |
| c | d |

$\pi_{A,C}(R)$

| A | B | C |
|---|---|---|
| a | b | c |
| c | b | d |

$\sigma_{B=b}(R)$

Existen algunas otras operaciones que se pueden usar en las relaciones que pueden ser expresadas con las cinco operaciones básicas . El ejemplo más simple es que cualquier sistema encerrado dentro de la diferencia está también encerrado dentro de la intersección, porque *R inter S* es equivalente a *R - (R - S)*. Así se puede usar la intersección como si fuera un operador básico del Álgebra relacional, sabiendo que se puede reemplazar por dos aplicaciones del operador de diferencia.

### 1.5.2 Leyes del Álgebra.

Así como otras álgebras, existen leyes que los operadores del Álgebra relacional obedecen.

La unión *(U)* es asociativa.

*[R U ( S U T ) = (R U S) U T]*

y conmutativa *[R U S = S U R]*, el producto cartesiano es asociativo, pero no conmutativo, porque el orden en que aparecen sus columnas es importante desde el punto de vista del conjunto de las listas.

Puede usarse la selección y la proyección para hacer cualquier pregunta natural acerca de relaciones simples.

Por ejemplo, si se tienen las relaciones MATERIALES(Nom_Prob,Nom_Mat,Precio), si quiere saberse qué materiales provee Juan, se puede decir:

$$\pi_{Nom_Mat}(\sigma_{Nom_Prob='Juan'}(\text{MATERIALES}))$$

Así, que, la selección enfoca en las tuplas que hablan acerca del proveedor 'Juan' y la proyección permite ver solamente el nombre de los materiales de esas tuplas. La expresión algebraica anterior, evalúa una relación de grado 1, y, su valor será una lista de todos los materiales que provee 'Juan'.

Si quiere saberse qué materiales provee 'Acme' y los vende a un precio menor que Q5.00 y los precios de cada artículo, se debe escribir:

$$\pi_{Nom_Mat,Precio}(\sigma_{Nom_Prob='Acme' \text{ y } Precio < 5.00}(\text{MATERIALES})).$$

## 1.6 Modelo de red.

Aproximadamente, el modelo de datos de red es el modelo de entidad relación restringido a ser binario. Esta restricción permite usar un simple grafo dirigido para el modelo de datos. En lugar del conjunto de entidades, el modelo de red habla de tipo de registros lógicos. Un tipo de registro lógico está formado por campos, donde existen lugares en donde los valores elementales tales como enteros y un grupo de caracteres, puede ser colocado. El conjunto de nombres, para los campos y su tipo, constituyen un formato de registro lógico.

Se puede suponer que existe una analogía cercana entre estos términos para redes y los términos usados para las relaciones, dentro de la correspondencia

| | |
|---|---|
| Formato lógico de registro | : Esquema de relación |
| Registro lógico | : Tupla |
| Tipo lógico de registro | : Nombre de la relación |

Sin embargo, existe una importante distinción entre tuplas de relaciones y registros de un tipo de registro. En el modelo relacional orientado a valores, las tuplas no son más que los valores de sus componentes. Dos tuplas con el mismo valor para los mismos atributos son la misma tupla. Por otro lado, el modelo de red está orientado a objetos, por lo menos en la extensión que soporta entidades de objetos. Los registros del modelo de red deben ser visualizados teniendo una llave invisible, que es en esencia la dirección del registro. Este único identificador sirve para hacer los registros distintos, aunque tengan los mismos valores en sus campos correspondientes. En efecto es factible tener tipos de registros que no tengan un solo campo.

La razón que da sentido para tratar registros que tienen identificadores únicos, independiente de los valores de sus campos, es que físicamente los registros contienen más datos que sólamente los valores en sus campos. En la construcción de una base de datos en el modelo de red existen apuntadores físicos a otros registros que representan las relaciones asociativas donde su tipo de registro está involucrado.

El conjunto de entidades se representan directamente por tipos lógicos de registros; los atributos de una entidad vienen a ser los campos del formato lógico del registro. El único caso especial es cuando una entidad *E* forma su llave con los campos de una entidad *F* donde *E* está relacionada a través de una relación asociativa *R*. No es necesario colocar estos campos de *F* en formato del registro de *E*, porque los registros de *E* no necesitan ser distinguidos por los valores de sus campos. Mejor dicho, ellos se distinguen por los apuntadores físicos localizados en los registros de *E* para representar la relación asociativa *R* y estos apuntadores no se llevan de un registro e de tipo *E* al registro correspondiente de tipo *F* que ocupa el valor de la llave para *e*.

La distinción entre las tuplas y registros dan una idea de la forma en que cada uno de los modelos, relacional y de red, sobresalen. El modelo relacional brinda la habilidad de usar los valores como componentes en formas arbitrarias y forman el camino que el diseñador inicialmente visualizó. En el Álgebra relacional se pueden hacer comparaciones entre los distintos componentes; sin embargo, no existe una forma verdadera para comparar estos componentes, sus lenguajes sólo permiten seguir sus apuntadores. Pero cuando en el modelo de red se siguen estos caminos tiene una mejor forma de expresarlos que el Álgebra relacional. Existe una importante ventaja y es que las relaciones sobre las acciones producen una relación y así se pueden construir, más fácilmente, expresiones complejas del Álgebra relacional. En el modelo de red, las operaciones no producen necesariamente una red o, por lo menos, una parte de ella.

## 1.7 Modelo jerárquico.

Una jerarquía es simplemente una red que es un grupo de árboles en donde todos los apuntadores se encuentran en dirección de los hijos a los padres.

Así como cualquier diagrama de entidad relación puede ser representado en el modelo de red y relacional, igual, un diagrama siempre se puede representar en el modelo jerárquico, sin embargo, para hacer esta transformación es necesario tipos de registros virtuales.

Para hacer la transformación, se debe iniciar en un nodo con una buena cantidad de apuntadores y hacer ese nodo la raíz del árbol, luego se deben unir todos los nodos que se puedan, tomando en cuenta que los apuntadores deben estar dirigidos a los padres. Lo mismo se debe hacer con otro nodo que permita hacer lo anterior. Eventualmente, cada nodo aparecerá en un árbol, una o más veces, y, en ese punto ya se tiene una jerarquía.

Las jerarquías que se puedan encontrar en los tipos lógicos de registros se encuentran en el nivel de esquema conceptual. Las instancias de la base de datos correspondiente al esquema consiste en un conjunto de árboles donde los nodos son registros , cada árbol es llamado registro de la base de datos. Un registro de la base de datos corresponde a un árbol del esquema de la base de datos, y, el registro raíz de un registro de base de datos corresponde a una entidad del tipo del registro raíz.

## 1.8 Lenguajes de bases de datos.

En los lenguajes ordinarios de programación las declaraciones y las sentencias ejecutables todas son partes de un lenguaje. En el mundo de las bases de datos. sin embargo, es común separar las dos funciones de declaración y cómputo en dos lenguajes distintos. La motivación es que, mientras que en un programa ordinario, los datos existen solamente mientras el programa se está ejecutando, en un sistema de bases de datos, los datos persisten y pueden ser declarados una sola vez. El trabajo se divide entre lenguajes de bases de datos especializados y ordinarios o lenguajes anfitriones.

### 1.8.1 Lenguajes de definición de datos.

El esquema conceptual es especificado en un lenguaje, proveído como parte de un DBMS, llamado lenguaje de definición de datos (DDL). Este lenguaje no es un lenguaje procedural, pero sí es un lenguaje que puede escribir los tipos de entidades y relaciones entre tipos de entidades, en términos de un modelo de datos particular.

**Ejemplo 1.10**: se puede definir una relación describiendo los vuelos de una línea aérea con la definición de datos:

```
CREATE TABLE VUELO(NUMERO:INT, FECHA:CHAR(6),ASIENTOS:INT,
      ORIGEN: CHAR(3), DESTINO:(3));
CREATE INDEX FOR VUELO ON NUMERO;
```

este código es un ejemplo del lenguaje de definición de datos de SQL. Las dos primeras líneas describen, sus atributos y su implementación física como caracteres y enteros de longitud fija. La tercera línea muestra que un índice sobre el número de vuelo será creado como parte del esquema físico, asumiendo que se harán búsquedas de información acerca de vuelos, dado su número, mucho más eficiente que si se tiene que hacer la búsqueda en el archivo completo de vuelos.

El lenguaje de definición de datos es usado cuando la base de datos es diseñada y se usa cuando el diseño es modificado, no se usa para obtener o modificar los datos en sí. Los lenguajes de definición de datos tienen sentencias que describen, en alguna forma abstracta, términos como los del ejemplo anterior, lo que físicamente la base de datos debería ser. El diseño detallado de la base de datos física se hace por las rutinas del DBMS que compilan sentencias en el lenguaje de declaración de datos.

La definición de subesquemas y su correspondencia al esquema conceptual, requiere un lenguaje de definición de datos para subesquemas, que es similar al mismo lenguaje de definición de datos. Algunas veces el lenguaje de subesquemas usa un modelo diferente al que usa el lenguaje de definición de datos.

### 1.8.2 Lenguajes de manipulación de datos.

Las operaciones sobre la base de datos requieren de un lenguaje especializado, llamado lenguaje de manipulación de datos (DML) o lenguaje de búsquedas donde se pueden expresar comandos para:

1. recuperar de la base de datos el número de asientos disponibles en el vuelo 999 en julio 24;

2. decrementar en 4 el número de asientos disponibles en el vuelo 123 en agosto 21;

3. encontrar todos los vuelos de GUA(Aeropuerto de Guatemala) a MEX(aeropuerto de México) en septiembre 19;

4. agregar el vuelo 675, con 150 asientos, de GUA a MEX en octubre 17.

Se ilustra entre (1) y (3) las búsquedas de la base de datos. El ejemplo (2) muestra una sentencia de actualización y puede ser implementado por un programa como las siguiente líneas de SQL:

```
UPDATE VUELO
SET ASIENTOS = ASIENTOS -4
WHERE NUMERO = 123 AND FECHA = 'AGOSTO 21';
```

en (4) se ilustra la inserción de un registro en la base de datos y puede ser expresado por un programa en SQL como:

```
INSERT INTO VUELO
      VALUES (675, 'OCTUBRE 17',150,'GUA','MEX');
```

El término lenguaje de búsquedas es frecuentemente usado como un sinónimo para lenguaje de manipulación de datos; estrictamente hablando, sólo algunas sentencias de un DML son búsquedas; son sentencias que extraen datos de la base de datos sin modificar la misma base de datos.

### 1.8.3 Lenguajes anfitriones.

Algunas veces la manipulación de la base de datos es hecha por un programa de aplicación, escrito, con el propósito de ejecutar alguna tarea. Es usualmente necesario para un programa de aplicación, hacer más que manipular la base de datos; debe ejecutar una variedad de tareas computacionales ordinarias. Por ejemplo, un programa usado por una línea aérea para hacer reservaciones no solamente necesita traer de la base de datos el número actual de asientos disponibles en el vuelo y actualizar ese número. Necesita tomar una decisión : ¿hay suficientes asientos disponibles? debería imprimirse el boleto, debe entrar en diálogo con el usuario, etc.

Así, los programas para manipular la base de datos están escritos en un lenguaje anfitrión, que puede ser un lenguaje de programación convencional como C o hasta COBOL. El lenguaje anfitrión es usado para decisiones,

desplegar preguntas y leer respuestas; es usado para todo, excepto para las búsquedas y modificaciones de la base de datos.

Los comandos de los lenguajes de manipulación de datos son invocados por un programa escrito en un lenguaje anfitrión, en una de dos formas, dependiendo de las características del DBMS.

1. Los comandos del lenguaje de manipulación de datos son invocados por la llamadas del lenguaje anfitrión; en procedimientos proveídos por el DBMS.

2. Los comandos son sentencias en un lenguaje que es una extensión del lenguaje anfitrión. Posiblemente existe un procesador que maneja las sentencias de manipulación de datos o un compilador puede manejar ambos, las sentencias del lenguaje anfitrión y las sentencias del lenguaje de manipulación de datos. Los comandos del lenguaje de manipulación de datos serán convertidos en llamadas a procedimientos, condicionado por el DBMS, así que la diferencia entre (1) y (2) no es muy grande

La siguiente figura muestra cómo el programa de aplicación interactúa con la base de datos. Existen *Datos locales* que corresponden al programa de aplicación; estos datos son manipulados por el programa en forma ordinaria. El programa de aplicación encaja con las llamadas, los procedimientos de la base de datos. Una búsqueda preguntando por datos causa la respuesta que es copiada de la base de datos a variables en el área de datos local; si existe más de una respuesta, entonces, las soluciones se recuperan una por una, cuando se hace la llamada por el programa de aplicación. Cuando se insertan o modifican datos, los valores son copiados de las variables de datos locales a la base de datos, siempre en respuesta a las llamadas de los procedimientos.

**1.9 Arquitectura de un sistema de bases de datos.**

En la siguiente figura se muestra un diagrama de cómo interactúan varios componentes de un administrador de bases de datos. En el lado derecho se observa el diseño o esquema de la base de datos, en la parte superior, el compilador del lenguaje de definición de datos, que produce una descripción interna de la base de datos. La modificación del esquema de la base de datos no es frecuente, comparado con la ejecución de búsquedas y otra manipulación de datos. En una base de datos multiusuario, esta modificación es normalmente responsabilidad del administrador de la base de datos, una persona o personas con responsabilidades con todo el sistema, incluyendo sus esquemas, vistas y autorización de accesar partes de la base de datos.

Se observa también el procesador del lenguaje de búsquedas, que brinda programas de manipulación desde dos fuentes; la primera, la búsqueda de los usuarios ingresadas directamente a una terminal; la segunda, los programas de aplicaciones dentro de un lenguaje anfitrión procesados para ser ejecutados posteriormente, talvez varias veces. Las porciones de un programa de aplicación escrito en un lenguaje anfitrión son manejadas por el compilador del lenguaje anfitrión, que no se muestra en la figura en discusión. Las porciones del programa de aplicación que son sentencias de un lenguaje de manipulación de datos son manejadas por el procesador del lenguaje de búsquedas, que se responsabiliza de la optimización de las sentencias. Se puede enfatizar que las sentencias de los lenguajes de manipulación de datos, especialmente las búsquedas, que extraen datos de la base de datos, se transforman, significativamente, con frecuencia, por el procesador de búsquedas, de tal manera que pueden ser ejecutados con más eficiencia que si

se ejecutaran como se escriben. Se observa al procesador de búsquedas, accesando las tablas de descripción, de la base de datos, que fueron creadas por el lenguaje de definición de datos, para averiguar algunos factores que se usan para la optimización de las búsquedas, así como la existencia de o inexistencia de ciertos índices.

Abajo del procesador de búsquedas se observa el administrador de la base de datos, cuyo rol es tomar los comandos del nivel conceptual y trasladarlos a comandos del nivel físico o nivel de archivos. El administrador de la base de datos mantiene y accesa las tablas de autorización y el control de la concurrencia. Las tablas de autorización, permiten al administrador de la base de datos chequear que el usuario tiene permiso de hacer la búsqueda o modificación que intenta hacer, sobre la base de datos. La modificación de la tabla de autorizaciones es hecha por el administrador de la base de datos.

Si el acceso concurrente a la base de datos, por diferentes búsquedas y manipulación , es permitido, el administrador de la base de datos mantiene la información necesaria en una tabla especial. Existen varias formas que se pueden hacer sobre la tabla de control de concurrencia. Por ejemplo, cualquier modificación a una relación puede generar un candado en la relación hasta que la modificación sea completada, previniendo así conflictos simultáneos de modificaciones.

El administrador de la base de datos traslada los comandos dados en operaciones sobre archivos, que son manejados por el administrador de archivos. Este sistema puede ser el propósito general del sistema de archivos proveído por el sistema operativo fundamental o puede ser un sistema especializado modificado para apoyar el DBMS.

## 1.10 Aplicaciones de sistemas de bases de datos.

La forma clásica de las bases de datos, fueron diseñadas para manejar una importante pero limitada clase de aplicaciones. Estas aplicaciones son como los que se sugieren: archivos de empleados, datos de una corporación, reservaciones de una línea aérea y registros financieros. Las características comunes de tales aplicaciones es que tienen grandes cantidades de datos, pero, las operaciones que se deben ejecutar sobre los datos son simples. En tales sistemas de bases de datos, las inserciones, borrar y recuperar registros específicos, predomina y la navegación entre pequeños números de relaciones o archivos, es una de las cosas más complejas que el sistema espera hacer.

La dicotomía de lenguaje anfitrión y lenguaje de manipulación de datos, es considerada generalmente como una ventaja, en lugar de ser una deficiencia en los sistemas de bases de datos. Por ejemplo, el poder limitado del lenguaje de manipulación de datos permite hacer una buena optimización de las búsquedas. Las mismas búsquedas escritas en un lenguaje de propósito general puede no ser optimizado en la misma forma radical por las técnicas conocidas. Sin embargo, existen algunas aplicaciones nuevas, de sistemas de bases de datos, que no siguen los paradigmas antiguos y en estas aplicaciones viene a ser importante la integración de los lenguajes anfitriones y los lenguajes de manipulación de datos.

## 1.11 Integración del lenguaje anfitrión y el de manipulación de datos.

Existen dos aproximaciones comunes para el problema de la combinación de la capacidad de rápido acceso con las capacidades de propósito general del lenguaje anfitrión.

1. La aproximación orientada a objetos es para usar un lenguaje con la capacidad de definición abstracta de tipos o clases. El sistema permite incluir al usuario, estructuras de datos para el rápido acceso en las clases donde sea necesario.

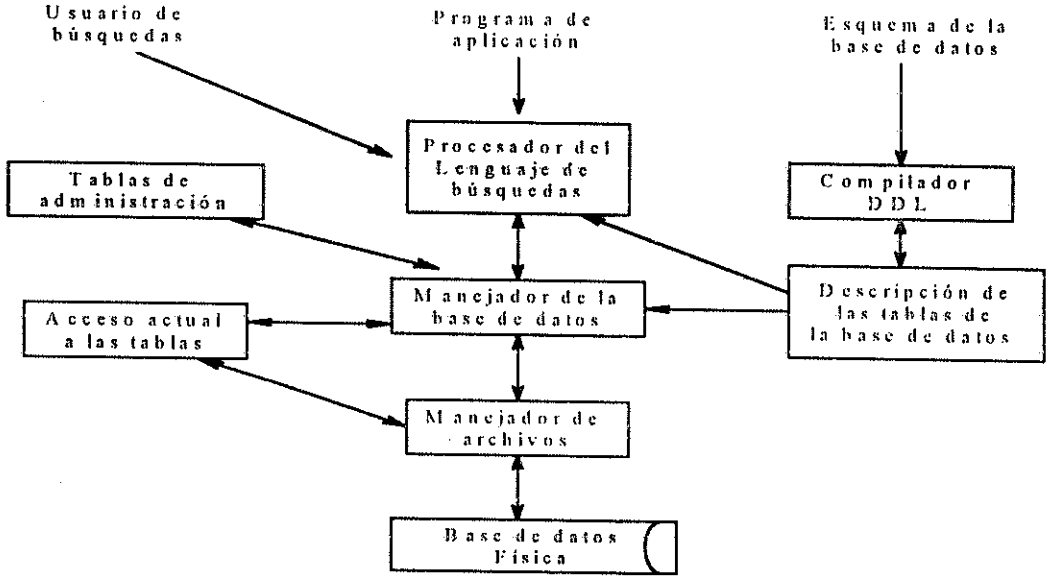



**Figura 1.4 Diagrama de un sistema de base de datos.**

2. La aproximación de la lógica usa un lenguaje que observa y procede de una forma parecida a las reglas de la lógica, si...entonces. Algunos predicados son considerados parte del esquema conceptual y se dan estructuras apropiadas para los índices mientras otras son usados para definir vistas, como si fueran parte de un subesquema. Otros pueden ser usados como si fueran parte de un simple programa de aplicación.

### 1.11.1 Lenguajes declarativos.

Existe una diferencia fundamental entre la aproximación orientada a objetos y la aproximación a la lógica, para diseñar un lenguaje integrado, el último es inherentemente declarativo y el primero no lo es. Recordando, un lenguaje declarativo es un lenguaje donde se puede expresar qué es lo que se quiere sin necesidad de explicar precisamente cómo se computará el resultado deseado. Un lenguaje que no es declarativo es procedural. Declarativo y procedural son términos relativos, pero es generalmente aceptado que los lenguaje ordinarios, como Pascal, C, Lisp y otros, son procedurales con el término declarativo, usado para lenguajes que requieren menor especificación considerando la secuencia requerida de pasos que hacen los lenguajes en esta clase.

El término declarativo de los lenguajes de búsquedas tiene una influencia significante sobre la arquitectura del sistema completo de la base de datos. Los siguientes puntos resumen las diferencias observadas entre los sistemas con lenguajes declarativos y procedurales, aunque se deba enfatizar que estas aseveraciones son generalidades que pueden ser contradichas por otros avances en la tecnología de las bases de datos.

1. Los usuarios prefieren lenguajes declarativos

2. Los lenguajes declarativos son más duros de implementar que los lenguajes procedurales porque los lenguajes declarativos requieren de una extensiva optimización por el sistema.

3. Aparece que la orientación a objetos y lenguajes declarativos son incompatibles.

## 1.12 Sistemas orientados a objetos.

El término sistemas administradores de bases de datos orientados a objetos, es usado para describir una clase de sistemas de programación con la capacidad de un DBMS y con un lenguaje combinado , anfitrión y de manipulación de datos, teniendo las siguientes características:

1. bjetos complejos, que es la habilidad de definir tipos de datos con una estructura anidada. Por ejemplo, una tupla es construida de tipos primitivos, enteros por ejemplo, por registros de información y una relación es construida de tuplas por un grupo de información, una relación es un conjunto de tuplas con un formato de registro particular. Se puede crear un registro cuyos componentes es un tipo, un conjunto de tuplas o estructuras más complejas;

2. encapsulación es la capacidad de definir procedimientos aplicados sólo a objetos de un tipo particular y la habilidad de requerir que el acceso a esos objetos es vía la aplicación de uno de esos procedimientos;

3. identidad de objeto, que significa la habilidad del sistema de distinguir dos objetos que parecen el mismo, en el sentido de que todos sus componentes de tipo primitivo son el mismo. Los tipos primarios o primitivos son, generalmente, caracteres, cadenas de caracteres, números y talvez otros cuantos tipos que tienen por naturaleza valores asociados.

Un sistema que maneja encapsulación y objetos complejos se dice que maneja tipos abstractos de datos o clases, esto es, un tipo abstracto o clase es una definición de una estructura junto con la definición de operaciones con las que el objeto o clase puede ser manipulada.

### 1.12.1 Identidad de objetos.

Para observar la diferencias entre los sistemas que manejan la identidad del objeto y los que no, considérese el siguiente ejemplo, donde se discute un archivo o relación consistente del par

EMPLEADO(NOMBRE,DEPARTAMENTO)

empleado-departamento. Si en el departamento de Redacción existen dos empleados con los nombre de Juan Pérez, si se piensa en la base de datos como

un archivo; simplemente se agregan dos registros en el archivo, cada uno tiene el primer campo con Juan Pérez y el segundo registro con Redacción. La notación del objeto es compatible con la identidad del objeto, porque la posición de un registro lo distingue de cualquier otro registro.

Sin embargo cuando se observan los datos como una relación, no se pueden almacenar dos tuplas, cada una con los valores de

("Juan Pérez","Redacción")

la razón es que, formalmente una relación es un conjunto, una tupla no puede ser un miembro de más de un conjunto. Es decir que no existe la posición de una tupla en relación en cuanto que los registros tienen una posición dentro de un archivo.

Un sistema que soporta la Identificación de objetos se referencia, generalmente, como Orientados a Objetos. Aunque ese término implique que puede manejar tipos de datos abstractos. Los sistemas que no manejan la Identidad de objetos son referenciados como orientados a valores u orientados a registros. Todos los sistemas basados en el modelo relacional de datos son orientados a valores, como lo son los sistemas basados en la lógica. Sin embargo, la mayoría de las primeras bases de datos fueron orientadas a objetos en el sentido limitado de manejar la identidad de objetos. Por ejemplo, los sistemas con modelo red son orientados a objetos en este sentido.

Pudiera parecer que la orientación a objetos es preferible que la orientación a valores, pero la preservación de la identidad de objetos no encaja con un sentido declarativo. Lo mismo pasa con la encapsulación, que es otra característica de los sistemas orientados a objetos.

## 1.13 Historia y perspectiva.

La siguiente figura muestra el desarrollo de los sistemas de bases de datos, desde sus orígenes alrededor de 1,960 hasta el presente. Se incorpora la década de los 90 donde los sistemas basados en la lógica inician a ser accesibles, además, se resumen tres puntos importantes de estos sistemas:

1. ya sea que son orientados a objetos u orientados a valores;
2. sea que soporten lenguajes declarativos para búsquedas;
3. y sea que separen el lenguaje de manipulación de datos del lenguaje anfitrión o que integren los dos.

| DÉCADA | SISTEMA | ORIENTACIÓN | DECLA. | DML/ANFITRIÓN |
|---|---|---|---|---|
| 1960 | Red y Jerárquico | Objetos | no | Separado |
| 1970 | Relacional | Valores | si | Separado |
| 1980 | OO-DBMS | Objetos | no | Integrado |
| 1990 | KBMS | Valores | si | Integrado |

**Figura 1.5 Pasado y futuro de los sistemas de bases de datos.**

Los primeros, verdaderos, manejadores de bases de datos, DBMS, aparecieron en la década de los 60 y fueron basados en modelos de red o jerárquicos, estos sistemas preveían eficiente acceso a cantidades masivas de datos, pero no tenían integrados los lenguajes de manipulación de datos y los lenguajes anfitriones tampoco preveían de lenguajes para búsquedas, que eran significativamente declarativos. Orientados a objetos en el sentido de que manejaban la identidad del objeto aunque no soportaban los tipos abstractos de datos.

En los años 70 se observa el advenimiento de los sistemas relaciónales . Una década de desarrollo fue necesaria, con muchas de las investigaciones dedicadas a las técnicas para la optimización de búsquedas necesarias para ejecutar los lenguajes declarativos que es una parte esencial de la idea relacional. Estos sistemas son declarativos y orientados a valores, pero no fácilmente permiten la integración de los lenguajes de manipulación de datos con los lenguajes anfitriones.

Se puede observar que los años 80 es la década de los sistemas orientados a objetos en el verdadero sentido del término, ya que soportan manejar la identidad del objeto como los tipos abstractos de datos. Son los primeros sistemas que proveen una buena integración entre los lenguajes de manipulación de datos y, los lenguajes anfitriones. Sin embargo, ellos representan un paso retrogrado ya que no son declarativos.

La tendencia es que a finales de los años 90 un verdadero KBMS sustituirá a los OO-DBMS, así como los sistemas relaciónales suplantaron a los primeros sistemas de bases de datos. Estos sistemas serán declarativos y proveerán la integración de los lenguajes anfitriones con los lenguajes de manipulación de datos. Se puede decir que serán inherentemente orientados a objetos y basados en la lógica . Hay que mencionar y que existe mucho por aprender acerca de la optimización de búsquedas antes de implementar un KBMS comercial.

# 2. LA LÓGICA COMO MODELO DE DATOS

Ahora inicia el estudio de la lógica en primer orden como una forma de representar el conocimiento y como un lenguaje para la representación de operaciones en las relaciones. Existe una jerarquía en los modelos de datos, cada cual con una representación de datos como la del modelo relacional, pero con un progreso más poderoso; lenguajes basados en la lógica para expresar las operaciones permitidas sobre los datos.

Se debe definir qué es el conocimiento y usar el término para referirse a sistemas con lenguajes declarativos basados en la lógica. Pero se debe distinguir entre sistemas de conocimiento y un administrador de sistemas basado en la lógica (Knowledge-base management system, KBMS). Un KBMS es un sistema que

1. provee lo que un DBMS provee: soporte eficiente para el acceso, el manejo de transacciones, elasticidad, etc. y

2. provee un simple, lenguaje declarativo que sirve para los roles jugados por los lenguajes de manipulación y los lenguajes anfitriones en un DBMS.

Un sistema de conocimiento, por otro lado, es un sistema que provee sólo (2); es un sistema de programación con un lenguaje declarativo. Sin embargo, existe una variedad de sistemas de conocimiento que son conocidos como sistemas expertos, lenguajes de sistemas de producción, lenguajes de programación de la lógica y otros nombres, estos sistemas de conocimiento no son de interés aquí. La orientación será hacia Prolog, probablemente el sistema de conocimiento mejor conocido, para proveer la notación para la lógica como conocimiento; tomando puramente el punto de vista declarativo considerando lo que las reglas lógicas significan aunque éstas se encuentren expresadas en la sintaxis de Prolog.

## 2.1 Notación de la lógica.

El tipo de razonamiento más sencillo realizado por el hombre implica la manipulación de proposiciones. Las siguientes son proposiciones simples denominadas fórmulas atómicas:

P1 : Fernando es un hombre;
P2 : Sandra es una mujer;
P3 : Fernando está casado con Sandra;
P4 : Fido es un perro;
P5 : Fido pertenece a Sandra;
P6 : Scamp es un perro.

Se pueden construir proposiciones más complejas a partir de combinaciones de proposiciones simples utilizando las palabras O, NO, SI...ENTONCES y Y. Por ejemplo:

P7 : René es un hombre O René es una mujer;
P8 : Juan NO es una mujer;
P9 : SI Fernando está casado con Sandra, ENTONCES Sandra está casada con Fernando;
P10: Fernando es un hombre Y Sally es una mujer.

Arbitrariamente, se pueden construir proposiciones complejas de esta manera. Hasta aquí, no se ha expresado cuáles de las proposiciones anteriores



PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

son verdaderas y cuáles falsas. La asignación de valores de verdad a las proposiciones puede declararse, explícitamente, de la siguiente manera:

A = {Fernando está casado con Sandra := Verdadero,
Scamp es un perro := Falso}

Donde los corchetes denotan un conjunto y el símbolo := significa tiene el valor. Es decir, una asignación de verdad es un conjunto de declaraciones, cada una de las cuales asigna un valor de verdadero o falso a una proposición.

Un método alternativo de especificar una asignación de verdad es hacer una lista de un conjunto equivalente de proposiciones, cada una de las cuales se supone que es verdadera (tales declaraciones se denominan asertos). Cualquier proposición P a la que se le asigne un valor de falso se escribe como aserto no P. Por ejemplo, el siguiente conjunto de asertos es equivalente a la anterior asignación de verdad A:

S1 = { Fernando está casado con Sandra, Scamp NO es un perro}

Existen algunas formas de razonar acerca de una asignación de verdad. Por ejemplo se podría ver si contiene una contradicción, es decir, si contiene una proposición a la que se ha asignado tanto el valor de verdadero como de falso. En tal caso, se dice que es inconsistente; si no, se dice que es consistente. Por ejemplo, S1 es consistente. Sin embargo, si se considera:

S2={ Juan NO es una mujer,
Juan es una mujer Y es alta }

Intuitivamente, se puede ver que S2 es inconsistente puesto que asigna tanto un valor de verdadero como de falso a una proposición. Ninguna de estas asignaciones está explícita en S2 pero puede hacerse tal cosa por la comprensión de las palabras NO y Y. Es decir:

a) Juan NO es una mujer es equivalente a:

Juan es una mujer := verdadero

b) Juan es una mujer Y Juan es alta, es equivalente a

{ Juan es una mujer := falso,
Juan es alta := verdadero}

2.1.1 Deducción.

Dada una asignación de verdad consistente, se puede emprender otra clase de razonamiento denominada deducción o inferencia. Por ejemplo, considérese el siguiente conjunto de asertos consistente:

{ Pedro es un Hombre O Pedro es una mujer,
Pedro NO es una mujer }

Aunque este conjunto no contiene el aserto "Pedro es un hombre" explícitamente, se puede inferir este aserto de la siguiente manera:

a) Pedro es un hombre O Pedro es una mujer significa que una de las asignaciones siguientes debe cumplirse:

Pedro es un hombre := verdadero
Pedro es una mujer := verdadero

b) Pedro NO es una mujer significa que la siguiente asignación no se cumple:

Pedro es una mujer := verdadero

c) De a) y b) se puede inferir que la siguiente asignación debe cumplirse:

Pedro es un hombre := verdadero

Considérese el siguiente conjunto de asertos:

S3 = { Pedro no es mujer,
Si Pedro está casado con María, ENTONCES María está casada con Pedro
Si María está casada con Pedro Y Pedro es progenitor de Luis ENTONCES Pedro no es un hombre O María es la madre de Luis,
Pedro tiene ojos azules Y Pedro es un hombre}

¿es S3 consistente? ¿Se puede deducir de S3 que María es la madre de Luis? Responder a estas preguntas puede llevar su tiempo y depende de la interpretación de lo que NO, SI... ENTONCES Y y O quieren decir.

Supóngase que se quiere escribir un programa que acepte el conjunto de asertos de S3 y que automáticamente dé respuestas. Hay que ser más formales con las definiciones de NO, SI...ENTONCES, Y y O. Y lo mismo hay que hacer con lo que se entiende por razonamiento. Se debe entonces definir cómo verificar la consistencia y cómo realizar la inferencia. Es aquí donde interviene la lógica formal. Dentro las muchas ramas de la lógica hay sistemas, cada uno de los cuales consta de :

a) un lenguaje bien definido para la representación del conocimiento;

b) métodos bien definidos para razonar

En esencia, un axioma deductivo es una regla con la que, dados ciertos hechos, puede deducir hechos adicionales. En una base de datos donde se quisiera almacenar lo que representan los axiomas fundamentales del sistema, se deben representar como tuplas y un grupo de reglas definidas.

### 2.1.2 Lenguajes de la lógica.

Los asertos acerca de partes del universo pueden formularse de forma precisa y clara mediante el empleo de lenguajes formales. Por ejemplo, considérese las siguientes fórmulas expresadas en un lenguaje de lógica proposicional:

a) (Pedro es un chico) V (Pedro es una chica)

b) (Pedro es un chico) -> ¬(Pedro es una chica)
lo anterior se lee como: (Pedro es un chico) implica no (Pedro es una chica)

c) (Juan es un chico) Λ (Marta es una chica)

La sintaxis de un lenguaje formal determina cómo pueden construirse declaraciones legales denominadas fórmulas bien formadas de ese lenguaje combinando componentes simples denominados fórmulas atómicas mediante el empleo de símbolos lógicos tales como

Λ y
V o
¬ no
-> implica

La sintaxis de un lenguaje consta de un conjunto de reglas que determinan qué fórmulas están bien formadas y cuáles no.

La semántica de un lenguaje formal determina cómo puede atribuirse significado a las fórmulas atómicas y cómo puede extenderse este significado para dar sentido a las fórmulas bien formadas en las cuales se encuentran las fórmulas atómicas.

En muchas lógicas, las fórmulas sólo pueden tener asignados valores de verdadero o falso. El efecto semántico de los símbolos lógicos está definido a menudo por la utilización de tablas de verdad. Por ejemplo, la siguiente tabla de verdad, en la cual T representa verdadero y F representa a falso, define el efecto de los símbolos lógicos de la lógica proposicional.

| A | B | A Λ B | A -> B | A V B | ¬A |
|---|---|---|---|---|---|
| T | T | T | T | T | F |
| T | F | F | F | T | T |
| F | T | F | T | T | T |
| F | F | F | T | F | T |

Tabla 2.1 Símbolos de la lógica.

A y B representan toda fórmula. Las definiciones de Λ, V y ¬ se ajustan al empleo corriente de las palabras Y, O y NO. Sin embargo, la definición de -> puede no estar de cuerdo con la idea de lo que significan SI...ENTONCES.

A pesar de ello, la definición de -> dada por esta tabla de verdad es la que maneja la lógica proposicional.

## 2.2 Reglas de inferencia.

Dado un conjunto de reglas bien formadas, S, es posible derivar o deducir consecuencias lógicas de S. En particular, es posible deducir consecuencias lógicas de S utilizando operaciones sintácticas solamente. Es decir, manipulando S estructuralmente, sin referencia a valores de verdad, es posible derivar nuevas fórmulas que garantizan que son secuencias lógicas de S. Los sistemas mediante los que se hace esto se denominan sistemas de deducción formal. Existen varios tipos de sistemas de deducción formal.

Un sistema axiomático consta de un lenguaje formal, un conjunto de reglas de inferencia y un conjunto de esquemas de axiomas lógicos.

Las reglas de inferencia definen operaciones sintácticas mediante las cuales pueden generarse nuevas fórmulas a partir de fórmulas dadas, es decir, permiten hacer deducciones.

Ejemplo 2.1: dadas las fórmulas bien formadas P y Q, se tienen las siguientes reglas de inferencia:

R1: De P -> Q y P, inferir Q. Esta regla se denomina modus ponens

R2: Si F es una fórmula válida universalmente, entonces, lo es F', donde F' se obtiene de F mediante la sustitución consistente de cualquier sub-fórmula. Por ejemplo, si B V ¬B es una fórmula válida universalmente, entonces lo es A V¬A. Esta regla se denomina regla de sustitución.

Los axiomas lógicos comprenden un conjunto de patrones para algunas de las fórmulas bien formadas válidas universalmente del lenguaje formal. El conjunto de axiomas lógicos de un sistema axiomático generalmente se elige tal que todas las fórmulas validas universalmente del lenguaje pueden generarse a partir de él, utilizando las reglas de inferencia del sistema axiomático.

La derivación de una fórmula F a partir de un conjunto de fórmulas S utilizando un sistema axiomático A se denomina deducción formal o prueba formal de F a partir de S en A. Una prueba formal de F desde S de A es una secuencia finita $F_1, F_2, \ldots, F_n$ de fórmulas tales que $F = F_n$, para cada i ($1 \leq i \leq n$)

a) $F_i$ es un axioma lógico de A.
b) $F_i \in S$, ($\in$ indica que pertenece al conjunto).
c) $F_i$ se genera a partir de fórmulas previas de la secuencia de acuerdo a una o más reglas de inferencia de A.

Modus ponens y sustitución son sólo dos de las diversas reglas de inferencia utilizadas en el razonamiento de cada día. Otras reglas incluyen la siguientes, donde P y Q representan cualquier fórmula bien formada:

R3: conjunción , de P y Q, inferir P Λ Q.

R4: simplificación, de P Λ Q inferir P.

R5: adición, de P, inferir P V Q

R6: transposición, de P -> Q, inferir ¬Q -> ¬P.

R7: silogismo hipotético, de P -> Q y Q -> P,

R8: silogismo disyuntivo, de P V Q y ¬P, inferir Q.

Los sistemas axiomáticos varían en la elección de reglas de inferencia y axiomas lógicos. Algunas reglas de inferencia están estrechamente relacionadas con axiomas lógicos particulares. Por ejemplo, la regla R4 y el axioma lógico [P Λ Q] -> P son equivalentes en el sentido de que el axioma lógico puede utilizarse para inferir P de P Λ Q cuando se emplea en conjunción con la regla modus ponens.

## 2.3 Teorías.

Una teoría consta de un sistema formal con un conjunto de fórmulas bien formadas que se sabe que son verdaderas en algún conjunto de interpretaciones propuesto. Estas fórmulas bien formadas se denominan axiomas propios de la teoría. Se da a continuación un ejemplo de un conjunto de axiomas propios:

S = { P1 : (Carlos es un hombre V (Carlos es una mujer)
P2 : ¬[(Carlos es un hombre Λ (Carlos es una mujer)]
P3 : (Pedro es un hombre) V (Pedro es una mujer)
P4 : ¬[(Pedro es un hombre) Λ (Pedro es una mujer)]
P5 : [(Carlos está casado con Pedro) Λ (Carlos es un hombre)] ->
(Pedro es una mujer)
P6 : [(Carlos está casado con Pedro) Λ (Carlos es una mujer)] ->
(Pedro es un hombre)}

Las interpretaciones pretendidas de una teoría son aquellas asignaciones de verdad que satisfacen todos los axiomas propios de la teoría de acuerdo con las reglas del sistema formal.

2.3.1 Demostración de teoremas.

Un teorema de una teoría es una fórmula que tiene una prueba formal de dicha teoría. El hecho de que F sea un teorema de una teoría se representa por:

T ⊢ F

Por ejemplo, supóngase que se tiene una teoría T'' que consta de los anteriores axiomas propios S junto con el siguiente esquema axiomático:

axiomas lógicos = { L1: [P -> Q] -> [¬P V Q],
L2: ¬[P Λ Q] -> [Q -> ¬P],
L3: [P Λ Q] -> [Q Λ P],
L4: [P V Q] V R -> [P V Q V R],
L5: ¬[P Λ Q] -> [¬P V ¬Q]}

reglas de inferencia = R1, modus ponens
R2, substitución
R7, silogismo hipotético

Lo siguiente es un teorema de la teoría T'':

[¬(Carlos está casado con Pedro) V ¬(Carlos es un hombre)
V ¬(Pedro es un hombre)]

La prueba se hace de la siguiente manera:

a) De P4, L2, R1, R2, inferir que:
F1: [(Pedro es un mujer) -> ¬(Pedro es un hombre)]

b) De P5, F1, r1, R7, inferir que:
F2: [[(Carlos está casado con Pedro) Λ (Carlos es un hombre)]
-> ([Pedro es un hombre)]

c) De F2, L1, R1, R2, inferir que:
F3: [¬[(Carlos esta casado con Pedro) Λ (Pedro es un hombre)] V
¬(Pedro es un hombre)]

d) De F3, L5, R1, R2, inferir que:

F4: [[¬(Carlos está casado con Pedro) ∧ ¬(Carlos es un hombre)] ∨ ¬(Pedro es un hombre)]

e) De F4, L4, R1, R2, inferir que:

F5: [¬(Carlos está casado con Pedro) ∨ ¬(Carlos es un hombre) ∨ ¬(Pedro es un hombre)]

## 2.4 Demostración de teoremas

Dada alguna teoría T, se puede demostrar si una fórmula F es o no un teorema de T utilizando repetidamente las reglas de inferencia del sistema formal de deducción formal y derivar por consiguiente nuevas fórmulas válidas.

Todos los métodos de demostración sintácticos empiezan con axiomas lógicos, axiomas propios y/o, suposiciones y/o y después aplican reglas de inferencia para intentar obtener la fórmula que hay que demostrar. Tales métodos se denominan deductivos sintéticos de encadenamiento hacia delante o conducidos por datos.

Los métodos de demostración de esta naturaleza dependen en gran medida de la inteligencia de la persona que realice la demostración. Cualquier intento para automatizar tales técnicas de demostración tendría que tener tal inteligencia programada en reglas de heurísticas o bien tendría que usar una estrategia de búsqueda ciega. Una búsqueda a ciegas implicaría que el sistema generará todos los teoremas que pudiera, confiando en encontrar el teorema requerido.

Ninguna de estas dos opciones es satisfactoria: en el primer caso, la mayoría de las heurísticas que se aplican cuando se resuelven problemas no están todavía bien definidas y aquellas que lo están, en general, sólo son aplicables en un dominio particular; en el segundo caso, incluso, con teorías relativamente triviales, el número de teoremas que se puede generar sería enorme y en muchos casos, infinito.

Los métodos analíticos son, en general, más mecánicos que los sintéticos y, como tales, resultan más manejables para la automatización.

Los métodos para realizar demostraciones que no se ajustan a un simple formato estándar, ya que las fórmulas que manipulan son irregulares, se denominan métodos no uniformes. Esta no uniformidad, complica la automatización de los métodos, como cabe suponer.

Debido a la interdefinibilidad de los operadores lógicos, es posible convertir fórmulas en formas regulares que impliquen sólo símbolos lógicos y reducir por ello el número de reglas de inferencia. Todas las fórmulas son convertidas a una forma particularmente regular llamada forma clausulada que solamente implica disyunción, conjunción y negación de subfórmulas. Una única fórmula de inferencia, llamada regla de inferencia de resolución se utiliza en las demostraciones. Por lo tanto, la resolución es particularmente apta para la automatización.

El método de resolución requiere que las fórmulas sean convertidas a una forma regular llamada fórmula clausada. Las fórmulas pueden ser convertidas una vez que se hayan transformado a una forma normal conjuntiva.

Una fórmula está en forma normal conjuntiva si es una conjunción de disyunciones de literales, donde una literal es una fórmula atómica o la negación de una fórmula atómica. Una fórmula está en forma normal disyuntiva si es una disyunción de conjunciones de literales.

Ejemplo 2.2: la forma normal conjuntiva de $\neg[[P \wedge Q] \vee R]$ es:

$$[\neg P \vee \neg Q] \wedge \neg R$$

La primera conjunción es la disyunción $\neg P \vee \neg Q$ y la segunda conjunción es la disyunción $\neg R$.

Método para convertir fórmulas a forma normal conjuntiva

a) -> Eliminación
Primero, todos los signos de implicación deben ser eliminados. Esto se hace realizando la siguiente conversión en la fórmula:

$\neg P \vee Q$ para $P \rightarrow Q$

b) Reducción del enlace de $\neg$
Ahora el alcance de los signos de negación se reduce para que cada ejemplificación de $\neg$ se aplique solamente a una proposición simple. Por ejemplo:

$\neg[P \vee Q]$ es $\neg P \wedge \neg Q$

$\neg[P \wedge Q]$ es $\neg P \vee \neg Q$

$\neg\neg P$ es $P$

c) Aplicación de un subconjunto de reglas distributivas

$P \vee [Q \wedge R]$ es $[P \vee Q] \wedge [P \vee R]$

$[P \wedge Q] \vee R$ es $[P \vee R] \wedge [Q \vee R]$

La lógica puede definirse de manera sucinta como un método formal de razonamiento. Como es formal, puede utilizarse para realizar tareas formales, como demostrar la validez de un argumento examinando sólo la estructura de ese argumento como una secuencia de pasos (o sea sin presentar atención al significado de esos pasos). En particular, desde luego, por ser formal se puede mecanizar, es decir, se puede programar y la máquina lo puede aplicar.

Ejemplo 2.3: considérese la siguiente fórmula:

$$[P \rightarrow Q] \rightarrow [\neg R \rightarrow [S \wedge T]]$$

los tres signos de implicación son eliminados en los pasos:

$\neg[P \rightarrow Q] \vee [\neg R \rightarrow [S \wedge T]]$
$\neg[\neg P \vee Q] \rightarrow [\neg R \rightarrow [S \wedge T]]$
$\neg[\neg P \vee Q] \vee [\neg\neg R \vee [S \wedge T]]$

Ahora se reduce el signo de negación:

$$\neg[\neg P \vee Q] \vee[\neg\neg R \vee[S \wedge T]]$$
$$[\neg\neg P \wedge \neg Q] \vee[\neg\neg R \vee[S \wedge T]]$$
$$[P \wedge \neg Q] \vee[R \vee[S \wedge T]]$$

Ahora se aplica la regla distributiva cuatro veces :

$$[P \wedge \neg Q] \vee [[R \vee S ] \wedge [R \vee T]]$$
$$[P \vee [[R \vee S] \wedge [R \vee T]]] \wedge [\neg Q \vee[[R \vee S] \wedge [R \vee T]]]$$
$$[P \vee [R \vee S]] \wedge [P \vee[R \vee T]] \wedge$$
$$[\neg Q \vee[R \vee S]] \wedge [\neg Q \vee [R \vee T].$$

## 2.5 Resolución.

Considérese ahora la siguiente disyunción de literales:

$$\neg P \vee \neg Q$$

Esta fórmula es equivalente a la fórmula:

$$\neg Q \vee \neg P$$

Es decir que el orden de los literales alrededor del operador lógico V es irrelevante, por lo tanto, una disyunción de literales puede considerarse como un conjunto de literales llamado cláusula, por ejemplo la siguiente cláusula es equivalente a la disyunción anterior:

$$\{\neg Q, \neg P \}$$

Esta cláusula puede leerse como no Q o no P.

Una cláusula que contenga como máximo una literal positiva se llama Cláusula de Horn. Los ejemplos siguientes son cláusulas de Horn:

$$\{P, \neg Q, \neg R\}$$
$$\{\neg P, \neg Q, R\}$$
$$\{\neg P, \neg Q, \neg R, \neg S\}$$
$$\{\neg P, Q, \neg R, \neg S, T\}$$

Considérese la siguiente fórmula:

$$[\neg Q \vee \neg P] \wedge \neg R$$

esta fórmula es equivalente a:

$$\neg R \wedge [\neg Q \vee \neg P]$$

Es decir que el orden de las subfórmulas alrededor de $\wedge$ es irrelevante. Por lo tanto, una fórmula en forma normal conjuntiva puede considerarse como un conjunto de cláusulas llamado conjunto clausulado. Por ejemplo, la fórmula anterior puede reemplazarse por el siguiente conjunto clausulado:

$$\{\{\neg Q, \neg P\}, \{\neg R\}\}$$

Las fórmulas de esta forma se dice que están en forma clausulada. Un conjunto clausulado, en el que todos los miembros sean Horn, se denomina conjunto clausulado Horn. El conjunto anterior es un ejemplo.

Puesto que se puede convertir cualquier fórmula proposicional a la forma normal conjuntiva, también se puede convertir a la forma clausulada. La ventaja de la forma clausulada es su uniformidad, que la hace ideal para el procedimiento automático.

Una asignación de $\alpha$ es apropiada a un conjunto clausulado S si cada fórmula atómica que aparezca en alguna cláusula de S está en el dominio de $\alpha$. $\alpha$ satisface un conjunto clausulado si y sólo si satisface todas las cláusulas del conjunto. $\alpha$ satisface una cláusula si y sólo si satisface uno de sus miembros. Por ejemplo, sea S el conjunto clausulado:

$\{\{P,Q\},\{\neg R\}\}$

Si $\alpha$ = ( P= verdadero, Q = falso, R= falso) entonces $\alpha$ satisface S.

Una cláusula o conjunto clausulado puede ser satisfacible, insatisfacible o una tautología de la misma manera que puede serlo una fórmula. Por ejemplo, el conjunto vacío es una tautología, pero la cláusula nula es insatisfacible, como lo es cualquier conjunto clausulado que contenga la cláusula nula.

Considérese el siguiente par de cláusulas:

C1: $\{P,\neg Q\}$ significando P o no Q
C2: $\{Q,S\}$ significando Q o S

ahora considérese la clausula C3: $\{P,S\}$ , que es una consecuencia lógica de C1 y C2, como se puede demostrar mediante la siguiente tabla de verdad:

| Q | S | C1:$\{P,\neg Q\}$ | C2:$\{Q,S\}$ | 3:$\{P,S\}$ |
|---|---|---|---|---|
| V | V | V | V | V |
| V | F | V | V | V |
| F | V | V | V | V |
| F | F | V | F | V |
| V | V | F | V | V |
| V | F | F | V | F |
| F | V | V | V | V |
| F | F | V | F | F |

Tabla 2.2 C3: $\{P,S\}$

Es decir , $\{P,S\}$ es verdadero en los cuatro casos en los que tanto $\{P, \neg Q\}$ como $\{Q,S\}$ son verdaderos.

Supóngase ahora que las fórmulas no estuvieran en forma clausulada sino que estuvieran escritas utilizando el operador lógico ->. Aparecerían de la siguiente manera:

C1: Q -> P
C2: $\neg$S -> Q

C3: ¬S -> P

Ahora está claro que C3 se deriva de C1 y C2, de ¬S -> Q y Q ->P , se infiere ¬S -> P.

Considérese ahora cómo deducir C3 de C1 y C2 sin tener que recurrir a construir tablas de verdad o convertir las cláusulas en alguna otra forma. La respuesta es muy directa. Se utiliza una regla de inferencia que reconozca la presencia de literales complementarias ¬Q de C1 y Q de C2. Esta regla llamada regla de inferencia de resolución, indica que se puede combinar C1 y C2 cancelando ¬Q en C1 con Q en C2 para formar una clausula nueva llamada resolvente de C1 y C2.

C1: { P, ¬Q }
C2: { Q, S }

C3: { P , S } el resolvente de C1 y C2.

Se puede generalizar y especificar la regla de inferencia de resolución más formalmente de la siguiente forma: de una cláusula C1 que contenga alguna literal ¬L y de una cláusula C2 que contenga un literal L, puede inferirse la cláusula:

C3: (C1 -{L} ) U (C2 - {¬L})

por ejemplo:

| C1 | C2 | C3: el resolvente de C1 y C2 |
|---|---|---|
| {P,Q,R} | {S,T,¬P} | {Q,R,S,T} |
| {P,Q} | {¬Q,S} | {P,S} |
| {T} | {¬T} | vacío |
| {P,Q,R,S} | {P,Q,¬R,S} | {P,Q,S} |

Por tanto, el resolvente de dos cláusulas es cualquier cláusula que se obtenga eliminando un par complementario de literales, uno de cada cláusula y uniendo las literales restantes en una cláusula única.

Si S es un conjunto cláusulado, entonces R está definido de la siguiente manera, donde i >= 1:

$R_0(S) = S$
$R_1(S) = S$ U todos los resolventes de S
$R_2(S) = R_1(S)$ U todos los nuevos resolventes de $R_1(S)$
$R_i(S) = R_{i-1}(S)$ U todos los nuevos resolventes de $R_{i-1}(S)$

R * S se puede definir, realizando el cierre de S bajo la operación de añadir todos los resolventes de las cláusulas a aquellas ya presentes, de la siguiente manera:

$R^*(S) = U\ R_j(S)$ donde $j>=0$ y $R_i(S) = R_{i+1}(S)$

Por ejemplo, considérese el siguiente conjunto clausulado:

S = {{P,Q} , {¬P,R} , {¬R,S}}

Aplicando la regla de resolución, se obtiene:

$R_0(S) = S$
$R_1(S) = S \cup \{\{Q,R\},\{\neg P,S\}\}$
$R_2(S) = R_1(S) \cup \{\{Q,S\}\}$
$R_3(S) = R_2(S) \cup 0$
$R^*(S) = \{\{P,Q\},\{\neg P,R\},\{\neg R,S\},\{Q,R\},\{\neg P,S\},\{Q,S\}\}$

La regla de resolución puede enunciarse de nuevo, de la siguiente manera:

Si S es un conjunto clausulado y D es un resolvente de dos cláusula en S, entonces, S y S U {D} son equivalentes.

## 2.6 Lógica de predicados.

La lógica proposicional permite razonar acerca de proposiciones. Sin embargo, no hay forma de razonar acerca de relaciones que existen entre entidades individuales de algún universo del discurso. Ni hay forma de expresar asertos generales que se apliquen a conjuntos de casos similares. En consecuencia, los lógicos han desarrollado una lógica expresiva llamada lógica de predicados.

Tal lógica puede utilizarse para describir y razonar acerca de partes del universo consideradas como estructuras relaciónales, donde una estructura relacional consta de un conjunto de entidades E, un conjunto de relaciones definidas en E y un conjunto de funciones definidas en E. Las entidades de un subconjunto E se conocen como entidades distinguidas.

Una definición más formal de una estructura relacional es que se trata de un cuádruple:

$$U = \langle E, N, R, H \rangle$$

donde:

- E es un conjunto de entidades $\{e_1, e_2, \ldots\}$ llamado dominio de U.
- N es un conjunto de entidades distinguidas $\{n_1, n_2, \ldots\}$ tal que N es un subconjunto de E.
- R es un conjunto de relaciones $\{r_1, r_2, \ldots\}$, cada una de las cuales está definida en E.
- H es un conjunto de funciones $\{h_1, h_2, \ldots\}$, cada una de las cuales está definida en E.

A continuación se da un ejemplo de una estructura relacional:

$U_1$:- E = El conjunto de todas las entidades que constituyen una Universidad.

N = {SUSANA, ROBERTO, VEINTE, TREINTA, VICTOR, CUARENTA, CINCUENTA, SESENTA}

R = {CASADO, EDAD}

H = {DOBLE}

donde:

```
CASADO = {<Susana, Roberto>,<Roberto,Susana>}
EDAD   = {<Susana, Veinte>,<Roberto,Treinta>,<Victor,Cuarenta>}
DOBLE  =  VEINTE  -> CUARENTA
          TREINTA -> SESENTA
```

Las relaciones y funciones de una estructura relacional pueden tener varios argumentos. Por ejemplo, CASADO tiene 2 argumentos ( dimensión 2), DOBLE tiene dimensión 1 y se llama función de un argumento.

## 2.7 Reglas de la lógica, utilizando la notación de PROLOG.

Es necesario dar una breve introducción de la notación de Prolog para las reglas de la lógica, dando una explicación más profunda en el siguiente capítulo. Para iniciar, las declaraciones de Prolog están compuestas por fórmulas atómicas, que consisten de un símbolo predicado aplicado, como si fuera el nombre de un procedimiento y argumentos. Estos argumentos deben estar formados por constantes (llamados átomos) variables y funciones de símbolos; estos últimos son aplicados a los argumentos como si se llamara a una función en un lenguaje de programación ordinario. Los símbolos predicados deben ser considerados de tal manera que producen verdadero o falso como resultados; son funciones de valores Booleanos. Las funciones de símbolos, por otro lado, deben ser considerados de tal manera que pueden producir valores del tipo que se deseen.

Siguiendo las convenciones de Prolog, los símbolos predicados, las funciones de símbolos y las constantes se inician con una letra en minúscula, con la excepción que las constantes también pueden ser valores enteros. Las variables deben iniciar con una letra mayúscula. Las declaraciones lógicas, generalmente, llamadas reglas, serán escritas de la forma: " si $A_1$ and $A_2$ and... $A_n$ are true, entonces, B is true", la sintaxis de Prolog para esta declaración es:

$B :- A_1 \ \& \ A_2 \ \& \ \ldots \ \& \ A_n.$

El símbolo ":-" puede leerse generalmente como "if". Note el punto final, que sirve como una marca de fin de la regla. Si n=0, entonces, la regla B se asegura incondicionalmente y se puede escribir como:

```
B.
```

Ejemplo: las dos siguientes reglas pueden ser interpretadas como una definición inductiva de la adición, se le atribuye el significado adecuado al símbolo sum y la función de símbolos s. Esto es , sum(X,Y,Z), es verdadero, cuando Z es la suma de X y Y, mientras s(X) es el sucesor de X esto es el entero que es una más que X . Entonces las reglas:

```
sum(X,0,X).
sum(X,s(y),s(Z)) :- sum(X<Y<Z).
```

esto quiere decir que X + 0 = X, y, que sí X + Y = Z, entonces X+(Y + 1)=(Z + 1).

Ejemplo: como el interés es sobre las reglas de la lógica expresando información de datos en una base de datos. Se puede apreciar la similitud entre la notación lógica de un predicado con sus argumentos y el nombre de una relación con sus atributos. Esto quiere decir que se puede que un predicado es verdadero para sus argumentos, si y sólo si esos argumentos forman una tupla de la relación correspondiente. Si se define la tabla EMPLEADOS, con

cuatro campos : nombre, departamento, salario y dirección; y, con esta tabla se puede definir la vista EMPLEADO-SEGURO con la regla lógica:

empleado-seguro(N,D,A) :- empleados(N,D,S,A)

La regla indica que para todos los empleados con nombre N, departamento D y dirección A, (N,D,A) es un factor del predicado empleado-seguro si existe un salario S, tal que (N,D,S,A) es un factor del predicado empleados. Notese que en general, una variable como S, que aparece del lado derecho del símbolo :-, pero no en el izquierdo se trata como existencialmente cuantificado; informalmente, cuando se lee la regla se dice que existe alguna S, después de decir el "si" que corresponde al símbolo ":-".

## 2.8 Expresividad de la lógica.

Se mencionó anteriormente que SQL y lenguajes similares, para la manipulación de datos, no tienen el suficiente poder para procesar cerraduras transitivas, tal como el manejo jerárquico, las reglas lógicas que usan las funciones de símbolos tienen todo el poder de una máquina de turing; pueden expresar cualquier computación que puede escribirse en un lenguaje de programación convencional. Las mismas reglas de la lógica sin las funciones de símbolos tienen el poder de expresar más computaciones que un lenguaje de manipulación de datos convencional.

Ejemplo 2.4: supóngase que se tiene la relación (o predicado) administra(E,M), que es verdadera exactamente cuando un empleado E reporta directamente al administrador M. Se debe definir otro predicado jefe(E,B), que será verdadero cuando B es administrador de E, o, su administrador del administrador o su administrador del administrador del administrador y así sucesivamente. jefe es la cerradura transitiva de administra. El predicado jefe puede ser expresado con las siguientes reglas como:

(1) jefe(E,M) :- administra(E,M).
(2) jefe(E,M) :- jefe(E,N) & administra (N,M)

El ejemplo anterior es un ejemplo típico de cómo las reglas de la lógica pueden ser usadas recursivamente, esto es, usadas para definir un predicado como jefe en términos de sí mismo. Para rebatir que una colección de reglas lógicas define algo en particular, se debe formalizar el significado de las reglas. Por el momento los siguientes argumentos intuitivos deben de dar idea de la semántica de las reglas de la lógica.

Primero, Cada vez que las reglas (1) y (2) indican que jefe(e,b) es verdadero, entonces, b es realmente lo que intuitivamente se considera como un jefe de e. Si se descubre jefe(e,b) por la regla (1), esto debe ser que administra(e,b) es verdadero, así seguramente b es un jefe de e, esto quiere decir que existe una cadena de manejo de 2 personas. Por otro lado, se debe considerar jefe(e,b) por la regla 2. Entonces, existe una c individual de tal manera que jefe(e,c) y también jefe(c,b) es un factor. Entonces, b está igualmente en la cadena de manejo abajo de e, así que la inferencia de que jefe(e,b) es correcta.

Se debe mostrar que cuando b está en la cadena de manejo abajo de e, se puede inferir jefe(e,b). La prueba es por inducción en la longitud de la cadena de manejo $e=c_1,\ldots,c_n=b$. Para la base, donde n=2, si b es el inmediato administrador de e, entonces, se puede hacer la inferencia por la regla (1). para la inducción, si existe una cadena de longitud mayor que la de e a b, digamos .

$e=c_1,\ldots,c_n=b$

entonces, por hipótesis inductiva, se puede inferir jefe($e,c_{n-1}$) y también dando administra($c_{n-1}$,b), así se usa la regla (2) para inferir jefe(e,b).

## 3. INTERPRETACIÓN DE REGLAS DE LA LÓGICA

En general, las reglas definen las instancias verdaderas de ciertos predicados, en términos de otros predicados que son definidos por relaciones de bases de datos, por ejemplo:

(1) jefe(E,M) :- administra(E,M).
(2) jefe(E,M) :- jefe(E,N) & administra (N,M).

*jefe* es el predicado y *administra* es la relación de la base de datos.

Existen tres formas de definir el significado de las reglas. En casos simples, como las del ejemplo anterior, todos estos métodos producen la misma respuesta. Como se permiten más clases complicadas de reglas lógicas, se pueden lograr diferentes aproximaciones que resulten en diferentes respuestas, porque las reglas lógicas, siendo declarativas por naturaleza, solo se sitúan en las propiedades del significado de la respuesta. En casos difíciles, no existe garantía que se defina una única respuesta o que exista una forma razonable de trasformar el programa declarativo en una secuencia de pasos que computen la respuesta.

### 3.1 Prueba teórica de la interpretación de las reglas

La primera de las tres interpretaciones que se puede dar a las reglas lógicas, es ésta, axiomas que son usados en una prueba. Esto es, de los datos de la base de datos, se observa que otros datos pueden ser probados usando las reglas en todas las posibles formas. Esta interpretación es la que se dio con las reglas del ejemplo 2.4 donde se mostró que los datos de jefe que pudieron ser probados de las reglas (1) y (2) arriba, unido a un grupo de datos de *administra* es exactamente lo que se puede esperar, si *jefe* da la interpretación "alguien arriba en la jerarquía de administración".

En casos simples como el del ejemplo 2.4, donde todos los axiomas son reglas si...entonces y no hay negaciones en las reglas o los datos, entonces, se sabe que todos los datos derivables usando las reglas, son derivables, aplicando las reglas como se hizo en ese ejemplo. Esto quiere decir que se utilizan las reglas sólo para substituir datos dados o probados en el lado derecho y, así, de ese modo, probando los datos resultantes del lado izquierdo. Esto cambia cuando hay negaciones, el conjunto de datos probables algunas veces no son lo que intuitivamente se quiere como significado para las reglas lógicas. Así que se debe definir la prueba teórica del significado como una colección de reglas que serán un conjunto de datos derivables de los datos dados o datos de la base de datos, usando las reglas sólo en dirección progresiva , esto es, infiriendo los lados izquierdos (consecuentes o conclusiones) de los lados derechos( antecedentes o hipótesis).

### 3.2 Modelo teórico de la interpretación de las reglas

En este punto de vista, se definen las reglas como un conjunto de mundos o "modelo". Una interpretación de una colección de predicados asignando falso o verdadero a cada posible instancia de esos predicados, donde los argumentos de los predicados se escogen de un dominio infinito de constantes. Usualmente, una interpretación está representada por su conjunto de instancias verdaderas. Para ser un *modelo* de un conjunto de reglas, una interpretación debe hacer la reglas verdaderas, no importando qué asignación de valores del dominio es hecho para las variables de cada regla.

**Ejemplo 3.1:** considérese las reglas

```
(1) p(X) :- q(x).
(2) q(x) :- r(x).
```

y suponga que el dominio de interés, son todos los enteros. Estas reglas indican que cuando *r* es verdadera para un entero dado entonces *q* es también verdadera y cuando *q* es verdadera, entonces *p* también lo es.

Un posible modelo, $M_1$, hace verdadero a *r(1),q(1),q(2),p(2)* y *p(3)* y hace *p*, *q* y *r falso* para todos los otros argumentos. Para ver que $M_1$ es un modelo, nótese que cuando se substituye $X = 1$ en la regla (1), ambos, el antecedente y el consecuente se hacen verdaderos y, así, la declaración " si *q(1)* entonces *p(1)*" es verdadero. De igual manera, si se substituye $X = 1$ en la regla (2), ambos lados son verdaderos. Cuando se substituye $X = 2$ en la regla (1) otra vez ambos lados son verdaderos, pero, cuando se substituye $X = 2$ en la regla (2) el antecedente es falso y el consecuente verdadero. Esta es otra forma de hacer una declaración si...entonces en verdadera y así la regla (2) se satisface. La misma observación aplica si se substituye $X = 3$ en la regla (1).

Cuando se substituye $X = 3$ en la regla (2) o se substituye cualquier valor además de 1,2 o 3 en cualquier regla, se obtiene una situación donde ambos, el antecedente y el consecuente son falsos. Otra vez la declaración si...entonces se hace verdadera. Así no importa qué substitución se haga, las reglas son verdaderas y, por lo tanto, se tiene un modelo.

Por otro lado, si se hace la regla *r(1)* verdadera y los tres predicados falsos para todos los otros valores, entonces no se tiene un modelo. La razón es que cuando se substituye $X = 1$ en la regla (2), se tiene un antecedente verdadero y un consecuente falso; y esa es la combinación que hace que la declaración si...entonces se haga falsa.

Cuando se usan las reglas para definir operaciones sobre una base de datos, se asume una instancia de un predicado de una base de datos de que es verdadera si y solo si la relación correspondiente ocupa esos datos en una tupla. Cuando se trata de extender la base de datos a un modelo sobre todos los predicados, se debe pensar en cualquier modelo como un posible mundo definido por las reglas. Por ejemplo, si se asume que *r* es un predicado de una base de datos en el ejemplo 3.1, mientras que *p* y *q* están definidos en términos de *r*. Se debe suponer también que *r(1)* es verdadera, mientras *r(X)* es falso para $X \neq 1$. Entonces el modelo

$$M_1 = \{r(1), q(1), p(1), q(2), p(2), p(3)\}$$

descrito en el ejemplo 3.1, es un posible mundo consistente en esta base de datos. Sin embargo, existe otro modelo consistente, $M_2 = \{r(1), q(1), p(1)\}$, en donde *p(1),q(1)*, y *r(1)* son verdaderas y todo lo demás es falso; ciertamente existe un numero infinito de modelos consistentes con una base de datos que tiene sólo *r(1)* verdadero. $M_2$ es especial, porque este es un modelo mínimo; no se puede cambiar un dato verdadero a falso y todavía tener un modelo consistente con la base de datos $\{r(1)\}$. Nótese que $M_1$ no tiene esta propiedad. Se puede cambiar *p(3)* y verdadero a falso en $M_1$ y todavía tener un modelo.

Además $M_2$ es el único modelo mínimo consistente con la base de datos $\{r(1)\}$. Este modelo es también lo que ocurre si se usa la prueba teórica para

la definición del significado de reglas. Esto es, empezando con las reglas (1) y (2) del ejemplo 3.1 y el valor *r(1)*, se puede probar *q(1)*, *p(1)* y no otro predicado. Estas coincidencia se observará que es verdadera para las reglas de "datalog" en general, ya que no involucran negación. Cuando la negación se permite como un operador, entonces no necesariamente debe existir un modelo mínimo único y ninguno de los modelos mínimos corresponde necesariamente al conjunto de valores que se pueden probar usando las reglas.

## 3.3 Definición computacional del significado.

La tercera forma de definir el significado de las reglas lógicas, es proveer un algoritmo para la ejecución de reglas, para que diga si un dato potencial es falso o verdadero (predicado con constantes para sus argumentos). Por ejemplo, Prolog define el significado de sus reglas utilizando esta forma, utilizando un algoritmo particular que involucra las búsquedas para pruebas de los datos potenciales. Desafortunadamente, el conjunto de datos para los cuales Prolog encuentra una prueba, esta manera no es necesariamente la misma que el conjunto de todos los datos para los que una prueba existe. Tampoco necesariamente el conjunto de datos que Prolog encuentra verdaderos son un modelo. Sin embargo, en algunos casos, Prolog tendrá buen éxito al producir el único modelo mínimo para un conjunto de reglas donde estas reglas se ejecutan como un programa de Prolog.

Otra aproximación para tratar las reglas, es de trasladarlas en secuencias de operaciones en álgebra relacional y para reglas datalog sin negación, se puede observar que el programa producido siempre ejecuta el único modelo mínimo y, por lo tanto, el conjunto de datos que pueden ser probados de la base de datos. Cuando la negación se permite se debe considerar un caso limitado llamado negación "estratificada" y entonces se debe observar que lo que nuestros programas producen es un modelo mínimo aunque no sea necesariamente el único modelo mínimo. Sin embargo, existe alguna justificación para seleccionar ese modelo mínimo entre todos los modelos mínimos posibles.

## 3.4 Comparación del significado.

Naturalmente, se debe preguntar cuál es la mejor forma de interpretación del significado para un programa lógico. Debería ser normal no tomar con seriedad el significado computacional de las reglas, pero para quienes desean implementar bases de datos basadas en la lógica la eficiente ejecución es esencial. No se puede usar las reglas lógicas como programas , a menos que se tenga una manera de procesar sus consecuentes y que sea eficiente.

Por otro lado, una definición puramente operacional del significado por reglas, tampoco es aceptable. No se tiene una preferencia entre la prueba teórica y el modelo teórico del significado, tanto como este significado es razonablemente claro para los usuarios de los lenguajes basados en la lógica. En la práctica parece que la aproximación del modelo teórico permite manejar más poderosamente ciertas clases de reglas que la aproximación de la prueba teórica. Cualquier interpretación que se escoja es esencial que muestre su equivalencia a un procesamiento apropiado del significado.

# 4. EL MODELO DE DATOS DATALOG.

## 4.1 Estructura básica.

Aquí se introduce la terminología básica necesaria para discutir el modelo de datos basado en la lógica llamado "datalog". El nombre "datalog" fue asignado para relacionar una versión de prolog adaptado para sistemas de bases de datos. Para trabajar con "datalog" se debe especificar que

1. Datalog no permite funciones de símbolos en sus argumentos. Por ejemplo, la función de símbolos usada en el ejemplo 2.4 no es permitida, se permiten solamente variables y constantes como argumentos de predicados.

2. El significado de los programas de datalog siguen el punto de vista del modelo teórico y cuando es equivalente, la aproximación de la prueba teórica. Sin embargo, Prolog tiene un significado computacional, pero en algunos casos se puede desviar al modelo teórico o a la prueba teórica del significado.

El modelo matemático de datos fundamental, para datalog es esencialmente el del modelo relacional. Los predicados de símbolos en datalog denotan relaciones. Sin embargo, como en la definición formal del álgebra relacional, estas relaciones no tienen atributos con los que pueden nombrar sus columnas. Mejor dicho, que son relaciones en el sentido del conjunto de listas, donde los componentes aparecen en un orden fijo y la referencia a una columna es sólo por la posición entre los argumentos de un predicado de símbolos dado. Por ejemplo, si $p$ es un predicado de símbolos, entonces, se puede referir a $p(X,Y,Z)$ y la variable $X$ denotará el primer componente de alguna tupla en la relación correspondiente al predicado $p$.

## 4.2 Predicados intencionales y extensionales.

Otra distinción entre el modelo relacional y el modelo datalog es que en datalog existen dos formas en las que las relaciones pueden ser definidas. Un predicado cuya relación es almacenada en la base de datos es llamada relación de base de datos extencional (EDB), mientras que una definida por reglas de la lógica es llamada relación de base de datos intencional (IDB), se asume que cada predicado de símbolos denota una relación EBD ó una relación IDB, pero no ambas.

En el modelo de datos relacional, todas las relaciones son relaciones EDB. La capacidad de crear vistas en los modelos como el modelo relacional es de alguna forma análogo a la habilidad de datalog de definir relaciones IDB. Sin embargo, la facilidad de la definición de vistas en DBMS relaciónales no es comparable en poder con el mecanismo de definición de las reglas de la lógica.

Los programas de Datalog se construyen de fórmulas atómicas, que son funciones de símbolos con una lista de argumentos, por ejemplo, $p(A_1,\ldots,A_n)$, donde $p$ es el símbolo del predicado. Un argumento en datalog puede ser una variable o una constante. Como ya se mencionó anteriormente, se asume que los nombres de predicados y constantes inician con un letra minúscula y los nombres de variables inician con una letra mayúscula; también se usan los números como constantes. Se debe asumir que cada predicado se encuentra



PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

asociado con un numero particular de argumentos y se debe usar $p^{(k)}$ para denotar un predicado de grado $k$ .

Una fórmula atómica denota una relación, es la relación de su predicado restringida por:

1. selección de igualdad entre una constante y el componente o componentes en donde aparece esa constante y
2. selección de igualdad entre componentes que tienen la misma variable.

Por ejemplo, considere la fórmula atómica

*clientes(Juan,Dirección,Balance)*

representa la relación $\sigma_{\$1=juan}$(CLIENTES). La fórmula atómica

*incluir(X,Item,X)*

representa $\sigma_{\$1=\$3}$(INCLUIR) que es la tupla donde el número de orden es igual al número de la cantidad ordenada.

Note que aunque no hay nombres para los atributos en el modelo datalog, la selección de variables sugestivas como *Dirección* ayudan a recordar que es lo que está sucediendo. Sin embargo, como en el álgebra relacional, se debe recordar el significado intuitivo de cada posición en una lista de argumentos.

### 4.3 Predicados construidos.

También se pueden construir fórmulas atómicas, con la comparación aritmética de predicados, =,≤, y así sucesivamente; estos predicados se identifican como predicados construidos. Las fórmulas atómicas con predicados construidos se escribirán en la notación usual infija, por ejemplo X< Y en lugar de <(X,Y). Otras fórmulas atómicas y sus predicados serán referenciadas como ordinarias cuando sea necesario hacer una distinción.

Los predicados construidos no representan necesariamente relaciones finitas. Podemos pesar en X < Y como la representación de la relación de todas la tuplas (x,y) tal que x < y, pero esta aproximación no es aceptable porque este conjunto es infinito y no está claro sobre qué dominio *X* y *Y* pueden agruparse. Por lo tanto, se requerirá que cuando una regla use fórmulas atómicas con un predicado construido, cualquier variable en esa fórmula esté limitada en un rango por alguna otra fórmula en el lado derecho de la regla. Por ejemplo, una variable debe estar limitada por la aparición en una fórmula atómica con un predicado EDB. Se puede encontrar entonces que las fórmulas atómicas construidas pueden ser interpretadas como selecciones en una relación simple o en la asociación de relaciones.

### 4.4 Cláusulas y cláusulas de Horn.

Una literal puede ser una fórmula atómica o una fórmula atómica negada; se denota como una fórmula atómica negada con $\neg p(A_1,\ldots,A_n)$, una fórmula atómica es una literal negada; una que no se encuentra negada es una literal positiva. Una cláusula es una suma ( lógica) de literales. Una cláusula de Horn es una cláusula con por lo menos una literal positiva. Se puede decir que una cláusula de Horn es:

1. una simple literal positiva, que se puede considerar como un dato;
2. una o más literales negativas, sin literales positivas, que es una integridad forzada;
3. una literal positiva y una o más literales negativas, que es una regla.

La razón por la que las cláusulas del grupo (3) son consideradas como reglas es que tienen una expresión natural como una inferencia. Por eso es que la Cláusula de Horn

$$\neg p_1 \ V \ldots V \ \neg p_n \ Vq. \qquad (4.1)$$

es lógicamente equivalente a $p_1 \ \hat{} \ldots \hat{} p_n \rightarrow q$. Para comprobar esto, nótese que si ninguna de las $p$ es falsa, entonces para hacer (4.1) verdadera, $q$ debe ser forzada a ser verdadera. Así, si $p_1, \ldots p_n$, son todas verdaderas, $q$ debe ser verdadera. Y si por lo menos una de las $p$ es falsa, entonces, no se debe forzar $q$; esta puede ser verdadera o falsa.

Para expresar la cláusula de Horn, se usará : $q$:- $p_1 \ \& \ldots \& p_n$. Para la clausal de Horn $p_1 \ \hat{} \ldots \hat{} p_n \rightarrow q$. Se llamará $q$ la cabeza de la regla y $p_1 \& \ldots \& p_n$
el cuerpo. Cada $p$ es considerada como una submeta. Una colección de cláusulas de Horn se determina como un programa lógico.

Se considera a las variables que aparecen en el cuerpo como existencialmente cuantificadas, mientras que otras variables son universalmente cuantificadas sobre la regla completa. Por ejemplo, la regla (1) de la siguiente figura dice "para todas la $X$ y $Y$, $X$ es hermano de $Y$ si existe $Z$ tal que $Z$ es padre de $X$ y $Y$, y $X$ y $Y$ no son el mismo individuo. Sin embargo, es correcto y lógicamente equivalente considerar todas las variables universalmente cuantificadas para la regla completa. Así, la regla (1) debe leerse, para todas la $X$, $Y$ y $Z$, si $Z$ es padre de $X$ y $Y$ y $X$ no es $Y$, entonces, $X$ es hermano de $Y$.

```
(1) hermano(X,Y) :- padre(X,Z) & padre(Y,Z) & X ≠ Y.
(2) primo(X,Y) :- padre (X,Xp) & padre(Y,Yp) & hermano (Xp,Yp).
(3) primo(X,Y) :- padre (X,Xp) & padre(Y,Yp) & primo (Xp,Yp).
(4) pariente(X,Y) :- hermano(X,Y).
(5) pariente(X,Y) :- pariente(X,Z) & pariente(Y,Z).
(6) pariente(X,Y) :- pariente(Z,y) & pariente(X,Z).
```

**Figura 4.1 Ejemplo de un programa de lógica**

## 4.5 Grafo de dependencias y recursividad.

Resulta necesario discutir, en un programa de lógica, la forma en que los predicados dependen uno del otro. Para hacer esto, se dibuja un grafo de dependencias, donde los nodos son predicados ordinarios. Existe una arista del predicado $p$ al predicado $q$, si existe una regla con una submeta donde su predicado es $p$ y con un encabezado cuyo predicado es $q$. Un programa de lógica es recursivo si el grafo de dependencias tiene uno o más ciclos. Note que un ciclo consiste en una arista que va de un nodo a él mismo y hace que el

programa sea recursivo, en la práctica se puede encontrar que los ciclos de un nodo son más comunes que los ciclos de nodos múltiples.

Todos los predicados que se encuentran en uno o más ciclos, se dice que son predicados recursivos. Un programa de lógica con un grafo de dependencias acíclico es no recursivo. Naturalmente, todos los predicados en un programa no recursivo son no recursivos, también se puede llamar predicado no recursivo a un predicado que se encuentre en un programa recursivo pero que no sea parte de ningún ciclo en el grafo de dependencias.

**Ejemplo 4.1:** supóngase que *padre* es una relación EDB y *padre(C,P)* significa que *P* es padre de *C*. Se define como relaciones IDB a *hermano*, *primo* y *pariente* de la figura 4.1. Hermanos son personas con un padre en común, pero se debe excluir la posibilidad de que se tenga *hermano(a,a)* como verdadero para cualquier individuo *a*, que explica la submeta *X* <> *Y* en la regla (1). Primo es gente con un antecesor común que tiene un mismo número de generaciones cada uno y, por lo menos, tiene dos generaciones de diferencia. No pueden ser hermanos o la misma persona.

Las reglas de la (4) al (6) definen *X* y *Y* que sean parientes si tienen un antecesor en común que no sea *X* ni *Y*. Esto es, la regla (4) dice que los hermanos son parientes en este sentido, ya que su padre común es un antecesor común que no puede ser el mismo para los dos. Entonces, las reglas (5) y (6) dicen que los parientes están relacionados con otros descendientes.

El grafo de dependencias de la figura 4.1 se muestra en la figura 4.2. Por ejemplo, la regla (1) incluye una arista de *pariente* a *hermano*. Nótese que no se usan nodos para los predicados construidos, como es el caso de ≠. La regla(2) justifica la existencia de una arista desde *pariente* a *primo* y una arista desde *hermano* a *primo*. La regla (3) justifica las aristas de *pariente* a *primo* y de *primo* a *primo* , la última arista es un ciclo e indica que el programa de lógica de la figura 4.1 es recursivo. Las aristas restantes están justificadas por las reglas (4)-(6).

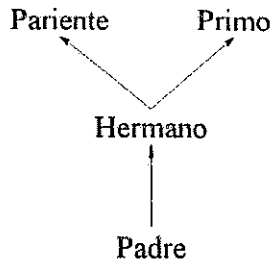



**Figura 4.2 Grafo de dependencias de la figura 4.1**

En la figura 4.1 existen dos ciclos, uno involucra sólo al *primo* y el otro involucra solamente a *pariente*. Así, estos predicados son recursivos y los predicados *padre* y *hermano* no son recursivos. Claro está que cada relación EBD así como *padre* debe ser no recursiva. En efecto, los predicados EDB son exactamente aquellos cuyos nodos no tienen arcos o aristas de entrada,

que implica que no pueden ser recursivos. El programa de la figura 4.1 es recursivo, porque tiene algunos predicados recursivos.

## 4.6 Reglas seguras.

Existen algunas operaciones que deben hacerse obligatoriamente sobre las reglas de datalog para que parezcan operaciones sobre relaciones finitas. Una fuente de infinidad es una variable que aparece solamente en predicados construidos. Otra es una variable que aparece sólo en la cabeza de la regla; el siguiente ejemplo ilustra el problema.

**Ejemplo 4.2:** La regla

```
masGrandeQue (X,Y) :- X > Y.
```

define una relación infinita, si X y Y se encuentran en el rango de los enteros o cualquier otro conjunto infinito.

Una simple aproximación para evitar reglas que crean relaciones infinitas de las finitas, es insistir que cada variable que aparece en la regla esté limitada. La idea intuitiva es que se asume que todos los predicados, no construidos, que aparecen en el cuerpo corresponden a relaciones finitas. Luego de asumir lo anterior, se debe asegurar que para cada variable *X*, existe un conjunto finito de valores $V_x$, tal que cualquier asignación de valor a las variables que hacen el cuerpo verdadero, el valor de *X* debe venir de $V_x$. Para definir formalmente las variables limitadas para una regla dada se menciona:

1. cualquier variable que aparece como un argumento en un predicado ordinario del cuerpo, es limitado;

2. cualquier variable X que aparece en una submeta $X = a$ o $a = X$, donde *a* es una constante, es limitada;

3. la variable *X* es limitada si ésta aparece en una submeta $X = Y$ o $Y = X$, donde *Y* es una variable que se conoce como limitada.

Nótese que (1) y (2) forman una base para la definición y (3) puede ser usada repetidamente para descubrir más variables limitadas.

Se define una regla como segura si todas sus variables son limitadas. La impresión crítica es que si las variables que aparecen en la cabeza y las variables que aparecen en las submetas con predicados construidos también aparecen en algunas submetas con un predicado ordinario, son igualados a constantes o son igualadas a otras variables limitadas mediante el uso recursivo de (3).

**Ejemplo 4.3:** la regla que se definió en el ejemplo 4.2 no es segura porque ninguna de sus variables es limitada.

En general, una variable que sólo aparece en la cabeza de una regla no puede ser limitada, por lo tanto, la regla no puede ser segura.

La regla (1) de la figura 4.1 es segura porque *X* ,*Y* y *Z* están limitadas por sus ocurrencias en las dos submetas de *pariente*; note que el predicado construido $X \neq Y$ no puede resultar en un número infinito de *hermanos*, porque *X* y *Y* se encuentran limitadas para ser individuos que aparecen en el primer

componente de la relación *pariente*. Las otras reglas de la figura 4.1 son de la misma manera seguras.

**Ejemplo 4.4**: considérese la siguiente regla

```
p(X,Y) :- q(X,Z)  &  W=a  &  Y=W
```

*X* y *Z* están limitadas por (1) (son argumentos) por la primera submeta en el cuerpo. *W* está limitada por la regla (2) (*a* es una constante) por la segunda submeta y, por lo tanto, (3) indica que *Y* está limitada por la tercera submeta, como todas las variables se encuentran limitadas la regla es segura.

## 5. ESTRATEGIAS DE EVALUACIÓN DE REGLAS.

En esta sección se tratará sólo con reglas seguras que no incluyan negación. Se podrá mostrar que existe un forma de convertir las reglas no recursivas a expresiones del Álgebra relacional; estas expresiones producen relaciones para los predicados IDB que son el modelo mínimo único de las reglas y el conjunto de datos IDB deducibles de las reglas y la base de datos. Se inicia con el punto de vista de la prueba teórica y se observará que el significado que se atribuye a las reglas es también el modelo mínimo único para las reglas.

### 5.1 Evaluación no recursiva de reglas.

Si las reglas no son recursivas, se pueden ordenar los nodos del grafo de dependencias $p_1,\ldots,p_n$ así que si existe una arista de $p_i$ a $p_j$ entonces $i<j$. Entonces se puede computar la relación para los predicados de $p_1,\ldots,p_n$ en ese orden, sabiendo que cuando se trabaja sobre $p_i$ las relaciones para todos los predicados que aparecen en los cuerpos de las reglas para $p_i$ ya han sido evaluados. La computación de la relación para $p_i$ se puede dividir en dos pasos.

1. Para cada regla $r$ con $p_i$ en la cabeza, computar la relación correspondiente al cuerpo de la regla. Esta relación tiene un componente para cada variable de $r$. Para computar la relación para el cuerpo de $r$, se debe tomar esencialmente la unión natural de la relaciones correspondientes a las submetas de $r$, tratando los atributos de estas relaciones como las variables que aparecen en la posición correspondiente de las submetas. Como las reglas no son recursivas, se debe asumir que las relaciones para las submetas ya fueron computadas.

2. Computar las relación para $p_i$ ella misma, en esencia, proyectando la relación para cada una de las reglas de $p_i$ en los componentes correspondientes a las variables de la cabeza y tomando la unión sobre todas las reglas con $p_i$ en la cabeza.

En cada uno de estos pasos, la computación es algo más complicado que las uniones y las proyecciones. Se debe tomar en cuenta las constantes que aparecen como argumentos y se debe considerar situaciones en donde una variable aparece en algunos argumentos de una submeta o de la cabeza.

#### 5.1.1 La relación definida por el cuerpo de una regla.

El primer paso es examinar el conjunto de valores que se deben substituir para las variables de una regla para hacer el cuerpo verdadero. En pruebas usando la regla, estas substituciones son exactamente las que permiten que se concluya que la cabeza, con la misma sustitución, sea verdadera. Por lo tanto se define la relación para una regla $r$, que tenga el esquema $X_1,\ldots,X_m$, en donde las $X_i$, son las variables del cuerpo de $r$, en algún orden seleccionado. Se quiere que esta relación tenga una tupla $(a_1,\ldots,a_m)$ si y sólo si, cuando se substituye $a_i$ para $X_i$, $1\leq i\leq m$ todas las submetas vienen a ser verdaderas.

Más, precisamente, supóngase que $p_1,\ldots,p_n$ es la lista de todos los predicados que aparecen en el cuerpo de la regla $r$, y, supóngase que $P_1,\ldots,P_n$ son relaciones donde $P_i$ consiste de todas estas tuplas $(a_1,\ldots,a_k)$ tal que

$p(a_1,\ldots,a_k)$, se sabe que es verdadero. Entonces, una submeta $S$ de la regla $r$ se hace verdadera por esta sustitución si se cumple con que:

i) Si $S$ es una submeta ordinaria, entonces $S$ se convierte en $p(b_1,\ldots,b_k)$, bajo esta sustitución y $(b_1,\ldots,b_k)$ es una tupla en la relación $P$ correspondiente a $p$.

ii) Si $S$ es una submeta construida, entonces, bajo esta sustitución $S$ se convierte en $b\Theta c$ y la relación aritmética $b\Theta c$ es verdadera.

**Ejemplo 5.1:** este es un ejemplo de cómo las relaciones para los cuerpos de las reglas son construidos. Considérese la regla (2) de la figura 4.1 supóngase que se tienen las relaciones $P$ y $S$ computadas para los predicados *pariente* y *hermano* respectivamente, supóngase también que existe una copia de $P$ con atributos $X$ y $X_p$ y otra con atributos $Y$ y $Y_p$ y que los atributos de $S$ son $X_p$ y $Y_p$, entonces, la relación correspondiente al cuerpo de la regla (2) es:

$$R(X,X_p,Y,Y_p) = P(X,X_p) \ |x| \ P(Y,Y_p) \ |x| \ S(X_p,Y_p) \qquad \text{(I)}$$

hay que recordar que |x| es un operador asociativo y conmutativo, así que el orden en que se agrupen los predicados es irrelevante.

Cuando se toman las uniones naturales es importante indicar que los nombres de los atributos corresponden a los componentes de cada relación. Finalmente, se debería apreciar la conexión cercana entre los nombres de atributos para los esquemas de relación y variables, en reglas lógicas.

Una forma equivalente de expresar la fórmula anterior es decir que se quiere una relación $R(X,X_p,Y,Yp)$ consistente de cada tupla $(a,b,c,d)$ tal que

1. (a,b) esta en $P$
2. (c,d) esta en $P$, y
3. (b,d) esta en $S$.

Esta relación es exactamente el conjunto de tuplas $(a,b,c,d)$ que cuando sustituidas por $(X,X_p,Y,Yp)$ en ese orden, hacen el cuerpo de la regla verdadero. Así, la fórmula anterior es la relación para el cuerpo de la regla (2) de la figura 4.1.

Si se considera la regla (1) de la figura 4.1, se necesita unir dos copias de $P$ y entonces seleccionar para la desigualdad aritmética $X \neq Y$. La expresión algebraica para la regla (1) es ésta:

$$Q(X,Y,Z) = \sigma_{X \neq Y}(P(X,Z) \ |x| \ P(Y,Z)) \qquad \text{(II)}$$

La relación $Q(X,Y,Z)$ computada por la expresión anterior consiste en todas las tuplas $(x,y,z)$ tal que :

1. $(x,z)$ esta en $P$,
2. $(y,z)$ esta en $P$, y
3. $x \neq y$.

Nuevamente, es fácil observar que estas tuplas $(x,y,z)$ son exactamente las que hacen el cuerpo de la regla (1) verdadera. Así que la fórmula (II) expresa la relación para el cuerpo de la relación para el cuerpo de la regla(1).

Considérese ahora:

```
p(X,Y) :- q(a,X) & r(X,Z,X) & s(Y,Z)                    (III)
```

supóngase que se tienen computadas la relaciones $Q$, $R$ y $S$ para las submetas $q$, $r$ y $s$, respectivamente. Se necesita construir una relación con un atributo $X$, conteniendo solamente el segundo componente de esta tupla ya que la primera submeta requiere aquellas tuplas de $Q$ que tienen el primer componente $a$. La relación que contiene solamente el segundo componente de esas tuplas, queda definida queda así :

$$T(X) = \pi_2(\sigma_{\$1=a}(Q))$$

también se debe restringir la relación $R$ ya que su primer y tercer componentes acarrean la variable $X$ en la segunda submeta, son iguales, entonces, se define

$$U(X,Y) = \pi_{1,2}(\sigma_{\$1=\$3}(R))$$

entonces la relación para el cuerpo de la regla (III) está definida por la expresión:

$$T(X) \ |x| \ U(X,Z) \ |x| \ S(Y,Z)$$

esta expresión define el conjunto de tuplas $(x,y,z)$ que hacen verdadero el cuerpo de la regla anterior, por ejemplo, el conjunto de tuplas $(x,y,z)$ tal que:

1. $(a,x)$ está en $Q$,
2. $(x,y,z)$ está en $R$, y
3. $(y,z)$ está en $S$.

En el siguiente algoritmo se describe como construir una expresión de álgebra relacional que compute la relación para el cuerpo de una regla.

**Algoritmo 5.1:** Computando la relación para el cuerpo de una regla, usando operaciones del álgebra relacional.

**Entrada:** el cuerpo de una regla datalog $r$, que se asume consiste en las submetas $S_1,\ldots,S_n$ involucrando las variables $X_1,\ldots,X_m$. Para cada $S_i = p_i(A_{i1},\ldots,A_{ik})$ con un predicado ordinario, existe una relación $R_i$ ya computada, done las $A$ son argumentos, variables o constantes.

**Salida:** una expresión de álgebra relacional, que se llamará

$$\text{REGLA-EVAL}(r,R_1,\ldots,R_n)$$

que computa de las relaciones $R_1,\ldots,R_n$ ( técnicamente no todas las relaciones $n$ deben estar presentes como argumentos porque algunas de las submetas pueden tener predicados construidos y así no tener relaciones correspondientes) una relación $R(X_1,\ldots,X_m)$ con todo y sólo las tuplas $(a_1,\ldots,a_m)$ tal que, cuando se sustituye $a_j$ por $X_j$, $1 \le j \le m$, todas las submetas $S_1,\ldots,S_n$ se hacen verdaderas.

**Método:** la expresión es construida por los siguientes pasos.

1. Para cada $S_i$ ordinaria, sea $Q_i$ la expresión $\pi_{Vi}(\sigma_{Fi}(R_i))$. donde $V_i$ es un conjunto de componentes incluidos, para cada variable $X$ que aparece entre los argumentos de $S_i$, exactamente un componente donde $X$ aparece. También $F_i$ es la conjunción (lógica ) de las siguientes condiciones:

   a) si la posición $k$ de $S_i$ tiene una constante $a$, entonces $F_i$ tiene el término $\$k=a$;

   b) si las posiciones $k$ y $l$ de $S_i$ ambas contienen la misma variable, entonces $F_i$ tiene el término $\$k=\$l$

   ( no es necesario agregar este término para todos los posibles pares $k$ y $l$, sólo para suficientes pares para las que todas las ocurrencias de la misma variable están forzadas a ser iguales . Por ejemplo, si $X$ aparece en las posiciones 2,5,9, y, 14 es suficiente agregar los términos \$2=\$5, \$5=\$9, y \$9=\$14.)

   Como un caso especial, si $S_i$ es de tal manera que no existen términos en $F_i$, por ejemplo, $S_i=p(X,Y)$, entonces, tomar $F_i$ para que sea la condición verdadera idéntica, así que $Q_i=R_i$.

2. Para cada variable $X$ que no aparezca entre las submetas ordinarias, computar una expresión $D_X$ que produce una expresión unaria conteniendo todos los valores que $X$ posiblemente pueda tener en una asignación que satisfaga todas las submetas de la regla $r$. Ya que $r$ es segura, existe alguna variable $Y$ a la que $X$ es igualada a través de una secuencia de una o más submetas, y, $Y$ está limitada por estar igualada a alguna constante $a$ en una submeta o por ser un argumento en una submeta ordinaria.

   a. Si $Y = a$ es una submeta, entonces, sea $D_X$ la expresión constante $(a)$.

   b. Si $Y$ aparece como el $j$-avo argumento de la submeta ordinaria $S_i$, sea $D_X$ igual a $\pi_j(R_i)$.

3. Sea $E$ la unión natural de todas las $Q_i$ definidas en (1) y las $D_x$ definidas en (2). En esta unión, se considera a $Q_i$ como una relación cuyos atributos son las variables que aparecen en $S_i$, y se considera a $D_x$ como una relación con atributo $X$.

4. Sea REGLA-EVAL$(r,R_1,\ldots,R_n)$ igual a $\sigma_F(E)$, donde $F$ es la conjunción de $X\Theta Y$ para cada submeta construida $X\Theta Y$ que aparece entre $p_1,\ldots,p_n$, y $E$ es la expresión construida en (3). Si no existen submetas construidas, la expresión deseada es simplemente $E$.

El ejemplo 5.1 ilustra la construcción de este algoritmo. Por ejemplo, la expresión $T(X) = \pi_2(\sigma_{\$1=a}(Q))$ es lo que se construyó por el paso (1) del algoritmo 5.1 de la primera submeta, $q(a,X)$, de la regla dada en (III); esto

es $T(X)$ es $Q_1$. Similarmente, $U(X,Z) = \pi_{1,2}(\sigma_{S1=S3}(R))$ en el ejemplo 5.1 es $Q_2$, construido de la segunda submeta, $r(X,Z,X)$. $Q_3$, construido de la tercera submeta, $S(Y,Z)$, es $S(Y,Z)$. No existen submetas construidas, así que no se necesitan dominios extras para ser construidos en el paso (2) y ninguna selección se necesita en el paso (4). Así, la expresión $T(X)$ $|x|$ $U(X,Z)$ $|x|$ $S(Y,Z)$ es la expresión final para el cuerpo de la regla (III).

El algoritmo 5.1 es correcto, en el sentido de que la relación $R$ producida tiene todas y sólo aquellas tuplas $(a_1,\ldots,a_m)$ tal que cuando se sustituye cada $a_j$ por $X_j$, cada submeta $S_i$ se hace verdadera.

### 5.1.2 Reglas restringidas.

Se consideran ahora cómo las relaciones para los cuerpos de las reglas están combinadas en relaciones para predicados. La idea básica es que se consideran todas la reglas con $p$ en la cabeza, computar la relaciones para estas reglas, proyectar dentro de las variables que aparecen en los encabezados y tomar la unión. Sin embargo, se tienen problemas cuando alguna de las cabezas con predicado $p$ tiene constantes o variables repetidas, por ejemplo $p(a,X,X)$. Se define entonces que las reglas para el predicado $p$ son restringidas si todos sus encabezados son idénticos y de la forma $p(X_1,\ldots,X_k)$ para variables distintas $X_1,\ldots,X_k$.

Es fácil restringir reglas, el truco es introducir nuevas variables para cada argumento de la cabeza del predicado e introducir submetas construidas en el cuerpo para dar fuerza a cualquier cosa que restrinja la cabeza del predicado anteriormente reforzado a través de constantes y la repetición de variables. Supóngase que se tiene una regla $r$ con la cabeza $p(Y_1,\ldots,Y_k)$, donde las $Y$ pueden ser variables o constantes con repeticiones. Se reemplaza el encabezado de $r$ por $p(X_1,\ldots,X_k)$, donde las $X$ son todas distintas, nuevas variables y a $r$ se le agregan las submetas $X_i=Y_i$ para todas la $i$. si $Y_i$ es una variable, entonces se debe de eliminar la submeta $X_i=Y_i$ y substituir $X_i$ por $Y_i$ en donde ésta se encuentre. Hay que hacer notar que cuando se hace una sustitución para $Y_i$ ya no se pueden hacer otras sustituciones para la misma variable $Y_i$.

**Ejemplo 5.2:** todas la reglas de la figura 4.1 ya se encuentran restringidas, considere entonces el predicado $p$ definido por las reglas

```
p(a,X,Y) :- r(X,Y).
p(X,Y,X) :- r(Y,X).
```

se restringen estas reglas, haciendo que los encabezados sean $p(U,V,W)$ y agregando submetas como se muestra :

```
p(U,V,W) :- r(X,Y) & U= a & V=X & W=Y.
p(U,V,W) :- r(Y,X) & U= X & V=Y & W=X.
```

si se substituye a $X$ y $Y$ por una de las nuevas variables $U$, $V$ o $W$ como es apropiado, entonces se obtiene:

```
p(U,V,W) :- r(V,W) & U =a.
p(U,V,W) :- r(V,U) & W =U.
```

esto es, en la primera regla, $X$ es reemplaza por $V$ y $Y$ es reemplazada por $W$, mientras que en la segunda , $X$ es reemplazada por $U$ y $Y$ es reemplazada por $V$.

Supóngase que $r$ es una regla y que el resultado de restringir $r$ es una regla $r'$, entonces:

a) si $r$ es segura entonces $r'$ es segura;

b) las reglas $r$ y $r'$ son equivalentes, en el sentido que, dadas las relaciones para los predicados de sus submetas, existe una solución para las variables de $r$ que hacen todas sus submetas verdaderas y hacen que su encabezado venga a ser $p(c_1,\ldots,c_n)$ si y sólo si existe alguna sustitución para las variables de $r'$ que hace que la cabeza de $r'$ venga a ser $p(c_1,\ldots,c_n)$.

### 5.1.3 Computando las relaciones para predicados no recursivos.

Una vez que se han rectificado las reglas, sólo queda hacer la proyección de la relación para cada cuerpo de la regla dentro de las variables de la cabeza y para cada predicado tomar la unión de los predicados producido de cada una de sus reglas.

**Algoritmo 5.2.** Evaluación no recursiva de reglas usando operaciones del álgebra relacional.

**Entrada:** un programa no recursivo de datalog y una relación para cada predicado EDB que aparece en el programa.

**Salida:** para cada predicado $p$ IDB, una expresión de álgebra relacional que da la relación para $p$ en términos de las relaciones $R_1,\ldots,R_m$ para los predicados EBD.

**Método:** se inicia rectificando todas las reglas. Después, construir el grafo de dependencias para el programa de entrada y ordenar los predicados $p_1,\ldots,p_n$, así que si el grafo de dependencias para el programa de entrada tiene una arista de $p_i$ a $p_j$, entonces, $i< j$. Se puede encontrar un orden ya que el programa no es recursivo y por lo tanto el grafo de dependencias no tiene ciclos. Entonces, para $i$=1,2,...,$n$, formar la expresión para la relación $P_i$, de $p_i$, como sigue:

si $p_i$ es un predicado EDB, sea $P_i$ , la relación dada para $p_i$. En caso opuesto, supóngase que $p_i$ es un predicado IDB. Entonces:

1. para cada regla $r$ teniendo $p_i$ como su cabeza, encuentre una expresión $E_r$ (usando el algoritmo 5.1), que compute la relación $R_r$ para el cuerpo de la regla $r$, en términos de relaciones para los predicados que aparecen en el cuerpo de $r$;

2. porque el programa no es recursivo, todos los predicados que aparecen en el cuerpo de $r$, ya tienen expresiones para sus relaciones en términos de las relaciones EDB. Sustituir la expresión apropiada para cada ocurrencia de una relación IDB en la expresión $E_r$ para obtener la nueva expresión $F_r$.

3. Renombrar las variables, si es necesario, se debe asumir que la cabeza de cada regla para $p_i$ es $p_i(X_1,\ldots,X_k)$. Entonces, tomar la expresión para $P_i$ para hacer la unión sobre todas las reglas $r$ para $p_i$, de $\pi_{X1,\ldots,Xk}(E_r)$.

**Ejemplo 5.3:** supóngase que se tienen las cuatro reglas:

```
(1)   p(a,Y) :- r(X,Y).
(2)   p(X,Y) :- s(X,Z) & r(Z,Y).
(3)   q(X,X) :- p(X,b).
(4)   q(X,Y) :- p(X,Z) & s(Z,Y).
```

aquí, $r$ y $s$ son predicados EDB y se supondrá que tienen sus relaciones dadas, $R$ y $S$. Los predicados $p$ y $q$ son predicados IDB, para los cuales se quiere computar relaciones $P$ y $Q$.

Se inicia rectificando las reglas, las reglas (1) y (3) necesitan modificación. Y el nuevo conjunto de reglas es:

```
(1)   p(X,Y) :- r(Z,Y) & X = a.
(2)   p(X,Y) :- s(X,Z) & r(Z,Y).
(3)   q(X,Y) :- p(X,b) & X =Y.
(4)   q(X,Y) :- p(X,Z) & s(Z,Y).
```

el orden para trabajar es: primero $p$ luego $q$, porque $q$ depende de $p$, pero no viceversa. La relación para el cuerpo de la regla (1) es, por el algoritmo 5.1, $R(Z,Y)\ |x|\ D_x(X)$, donde $D_x = \{a\}$; la relación para la regla (2) es $S(X,Z)\ |x|\ R(Z,Y)$. Ambas expresiones deben ser proyectadas sobre la lista de atributos $X$, $Y$ antes de tomar la unión. Como un caso especial la proyección no se necesita para la primera de éstas. Así, la expresión para P es:

$$P(X,Y) = \pi_{X,Y}(R(Z,Y)\ |x|\ \{a\}(X))\ \cup\ \pi_{X,Y}(S(X,Z)\ |x|\ R(Z,Y))$$

ahora, se considera $q$. La relación para la regla (3) se computa como sigue: por el algoritmo 5.1, la expresión para la submeta $p(X,b)$ es

$$\pi_X(\sigma_{Z=b}(P(X,Z)))$$

aquí, $Z$ es una variable arbitrariamente escogida que desaparece en la proyección. Esta expresión produce una relación solamente sobre el atributo $X$, se necesita una expresión que genere todos los posibles valores de $Y$, ya que $Y$ no aparece en otro lugar. Ya que $Y$ se iguala a $X$, se sabe que sólo los valores de $X$ pueden ser valores de $Y$, así que se puede tomar un argumento donde aparece $X$, nombrando el primer argumento de $P$ como el dominio para $Y$. Este dominio es el que se expresa por $\pi_Y(P(Y,W))$; $W$ es otra variable arbitrariamente escogida. Luego, se toma el producto cruz de la expresión para $p(X,b)$ con el dominio para $Y$, se selecciona para $X = Y$, por la submeta de la regla (3) $X = Y$. Así, la expresión para el cuerpo de la regla (3) es

$$\sigma_{X=Y}(\pi_X(\sigma_{Z=b}(P(X,Z)))\ \ x\ \ \pi_Y(P(Y,W)))$$

finalmente, la expresión para la regla (4) es $P(X,Z)\ |x|\ S(Z,Y)$ y la expresión para $Q$ es:

$$Q(X,Y) = \sigma_{X=Y}(\pi_X(\sigma_{Z=b}(P(X,Z)))\ \ x\ \ \pi_Y(P(Y,W)))\ \ \cup\ \ \pi_{X,Y}(P(X,Z)\ |x|\ S(Z,Y))$$

técnicamente se debería sustituir primero para $P$ la expresión que se construyó para $P$, en orden de obtener $Q$ en términos, solamente, de las relaciones $R$ y $S$ de la base de datos.

El algoritmo 5.2 computa correctamente la relación para cada predicado, en el sentido de que la expresión que construye para cada predicado IDB produce ambos:

1. el conjunto de valores para el predicado que puede ser probado de la base de datos, y

2. el único modelo mínimo de las reglas.

## 5.2 Computando el significado de reglas recursivas.

El algoritmo 5.2 no se aplica a programas recursivos de datalog, porque no existe un orden para los predicados que permite aplicar el algoritmo. Cuando existe un ciclo en el grafo de dependencias, el primer predicado en ese ciclo que se trata de evaluar tendrá una regla con una submeta cuya expresión no se encuentra disponible.

Sin embargo, la aproximación de la prueba teórica aún tiene sentido si se recuerda que es permitido derivar algunos datos usando una regla y más tarde usar nuevos datos derivados en el cuerpo de la regla para derivar más datos. Si se inicia con una base de datos finita y se usan sólo tipos de reglas como las de datalog, entonces, existe sólo un número finito de datos diferentes que posiblemente pueden ser derivados; estos deben ser de la forma $p(a_1,\ldots,a_k)$, donde $p$ es un predicado IDB mencionado en las reglas, y $a_1,\ldots,a_k$ son constantes que aparecen en la base de datos.

Considérese un programa de datalog con las relaciones EBD $R_1,\ldots,R_k$ dadas y con las relaciones IDB $P_1,\ldots,P_m$, para ser computadas. Para cada $i$, $1\leq i\leq m$, se puede expresar el conjunto probable de valores para el predicado $p_i$ (correspondiente a la relación $P_i$) por la asignación:

$$P_i := \text{EVAL}(p_i, R_1, \ldots, R_k, P_1, \ldots, P_m)$$

donde EVAL es la unión de REGLA-EVAL (como se definió en el algoritmo 5.1) para cada regla de $p_i$, proyectada sobre las variables de la cabeza. Si se inicia con todas las $P_i$ vacías y se ejecuta una asignación tal como estas para cada $i$, repetidamente, eventualmente se encontrará un punto donde no existan más valores que puedan ser agregados a alguna de las $P_i$. Ahora el símbolo de asignación viene a ser igual, el conjunto de los valores IDB que pueden ser probados, satisfacen la ecuación

$$P_i = \text{EVAL}\ (p_i, R_1, \ldots, R_k, P_1, \ldots, P_m)$$

para todas las $i$. Se llamarán ecuaciones de datalog ya que son ecuaciones derivadas de un programa de datalog.

**Ejemplo 5.4:** las reglas de la figura 4.1 pueden ser vistas como las siguientes ecuaciones. Se usa $P$, $S$, $C$ y $P$ para las relaciones correspondientes a *padre*, *hermano*, *primo* y *pariente* respectivamente.

$$S(X,Y) = \pi_{X,Y}(\sigma_{X\neq Y}(P(X,Z) \ |x| \ P(Y,Z)))$$

$$C(X,Y) = \pi_{X,Y}(P(X,X_p) \ |x| \ P(Y,Y_p) \ |x| \ S(X_p,Y_p)) \quad U$$

$$\pi_{X,Y}(P(X,X_p) \ |X| \ P(Y,Y_p) \ |x| \ C(X_p,Y_p))$$

$$R(X,Y) = S(X,Y) \ U \ \pi_{X,Y}(R(X,Z) \ |x| \ P(Y,Z)) \ U \ \pi_{X,Y}(R(Z,Y) \ |x| \ P(X,Z))$$

### 5.2.1 Puntos fijos de las ecuaciones de datalog.

El reemplazamiento del símbolo 'si', :-, en las reglas de datalog por una igualdad, para formar las ecuaciones datalog está justificado ya que el significado de las reglas no es más ni menos que lo que se puede probar usando las reglas. Sería adecuado que existiera una única solución para el conjunto de ecuaciones, pero generalmente existen muchas soluciones. Para un conjunto dado de relaciones para los predicados EDB, $R_1,\ldots,R_k$, un punto fijo de las ecuaciones datalog, respecto de $R_1,\ldots,R_k$, es una solución para las relaciones correspondientes a los predicados IDB de esas ecuaciones.

Un punto fijo $P_1,\ldots,P_m$, respecto de las relaciones EDB dadas $R_1,\ldots,R_k$, junto con estas relaciones, forman un modelo de las reglas de donde las ecuaciones datalog provienen. Para probar esto, sea $M$ el modelo en donde sólo los datos que son tuplas en $P_1,\ldots,P_m$ y $R_1,\ldots,R_k$, son verdaderas. Entonces, cualquier asignación de constantes a las variables que hacen verdadero el cuerpo de alguna regla $r$, deben hacer verdadera también la cabeza de $r$. Si se dice que esa cabeza es $P(a_1,\ldots,a_n)$, entonces, la tupla $(a_1,\ldots,a_n)$ debe estar en la relación para el predicado IDB $p$, sino las relaciones IDB escogidas no son un punto fijo de las ecuaciones.

Sin embargo, no es verdadero que cada modelo de un conjunto de reglas sea un punto fijo de las ecuaciones correspondientes, porque el modelo puede tener muchos datos y algunos de ellos aparecen en los lados izquierdos de las ecuaciones pero no en el lado derecho. Por otro lado, los más importantes son los puntos fijos y los modelos que son mínimos en el sentido que no tienen un subconjunto propio de valores que también es un punto fijo. Las porciones IDB de modelos mínimos son siempre puntos fijos, y ,en efecto, puntos fijos mínimos.

Sean las variables de las ecuaciones $P_1,\ldots,P_m$ correspondientes a los predicados IDB $p_1,\ldots,p_m$ y póngase mayor atención en particular en las relaciones $R_1,\ldots,R_k$ asignadas a los predicados EDB $r_1,\ldots,r_k$. Una solución o punto fijo, para las relaciones EBD $R_1,\ldots,R_k$ asignar a $P_1,\ldots,P_m$ las relaciones particulares $P^{(1)}{}_1,\ldots,P^{(1)}{}_m$ tal que las ecuaciones sean satisfechas. Si $S_1=P^{(1)}{}_1,\ldots,P^{(1)}{}_m$ y $S_2=P^{(2)}{}_1,\ldots,P^{(2)}{}_m$ son dos soluciones para un conjunto dado de ecuaciones, se dice que $S_1 \leq S_2$ si la relación $P^{(1)}{}_i$ es subconjunto de la relación $P^{(2)}{}_i$ para todas las $i$, $1 \leq i \leq m$. Entonces $S_o$ es el punto fijo mínimo para un conjunto de ecuaciones respecto de las relaciones EDB $R_1,\ldots,R_k$, si para cualquier solución $S$, se tiene $S_o \leq S$. Más, generalmente, $S_o$ es un punto fijo mínimo si no existe otro punto fijo $S$ tal que $S \leq S_o$. Hay que notar que si existe un punto fijo más pequeño, entonces es el único punto fijo mínimo. Sin embargo, pueden existir algunos puntos fijos mínimos que no son comparables por $\leq$, y ,en ese caso, no existe punto fijo más pequeño.

Para mencionar un ejemplo, considérese el problema de computar la cerradura transitiva de un grafo dirigido. Si el grafo es representado por un predicado EDB *arc* tal que $arc(X,Y)$ es verdadero si y sólo si existe un arco o arista desde el nodo $X$ al nodo $Y$, entonces, se puede expresar el camino en el grafo por las reglas:

(1) camino(X,Y) :- arc(X,Y)
(2) camino(X,Y) :- camino(X,Z) & camino(Z,Y)

La primera regla indica que el camino puede ser un simple arco y la segunda indica que la concatenación de cualquiera dos caminos, uno desde *X* a *Z* y otro desde *Z* a *Y*, produce un camino desde *X* a *Y*. Este par de reglas no es necesariamente la mejor forma en que se pueden definir caminos, pero son, probablemente, la forma más natural.

Se pueden transformar estas reglas en una simple ecuación para la relación *P* que corresponde al predicado camino. La ecuación asume que existe una relación dada *A* correspondiente al predicado *arc*.

$$P(X,Y) = A(X,Y) \cup \pi_{X,Y} (P(X,Z) \mid x \mid P(Z,Y))$$

Supóngase que los nodos son {1,2,3} y *A* representa los arcos 1->2 y 2->3 que es $A = \{(1,2),(2,3)\}$. La primera regla para *camino* indica que (1,2) y (2,3) están en *P*, y la segunda regla implica que (1,3) esta en *P*. Sin embargo, no se requiere deducir la existencia de más caminos ya que $P = \{(1,2), (2,3), (1,3)\}$ es una solución de la ecuación,

$$\{(1,2), (2,3), (1,3)\} = \{(1,2), (2,3)\} \cup \pi_{X,Y} (\{(1,2), (2,3), (1,3)\} \mid x \mid \{(1,2), (2,3), (1,3)\})$$

es una igualdad. Interpretando lo anterior, se debe recordar que el operando izquierdo de la unión es una relación sobre los atributos *X,Z*, y, su operando derecho es una relación sobre los atributos *Z*, *Y*. Así, la expresión $\pi_{X,Y}(P(X,Z) \mid x \mid P(Z,Y))$ puede ser pensada como la composición de la relación *P* con ella misma y su valor aquí es $\{(1,3)\}$.

La solución es la prueba teórica del significado de las reglas, porque se deriva de una relación *A* EDB, exactamente lo que las reglas permiten probar. También es fácil de observar que es el modelo mínimo de reglas o el punto fijo más pequeño de la ecuación anterior, respecto de la relación dada *A*, porque cada dato derivado se puede mostrar que se encuentra en cada modelo o punto fijo conteniendo la relación extensional *A*.

Sin embargo, existen otras soluciones para la ecuación. Supóngase, que (1,1) también existía en P. Las reglas no implican más caminos dado que $A=\{(1,2),(2,3)\}$ y $P=\{(1,1),(1,2),(2,3),(1,3)\}$. Nótese como (1,1) se prueba a sí mismo si hace que $X = Y = Z = 1$ en la regla (2). Así, otra solución para la ecuación es:

$$\{(1,1),(1,2),(2,3),(1,3)\} = \{(1,2),(2,3)\} \cup \pi_{X,Y}(\{(1,1),(1,2),(2,3),(1,3)\} \mid x \mid \{(1,1),(1,2),(2,3),(1,3)\})$$

Similarmente, se puede definir a *P* como el conjunto de los 9 pares $(i,j)$ donde $1 \le i,j \le 3$, y, ese valor también satisface la ecuación, Por otro lado, no todos los valores de *P* satisfacen la ecuación. Por ejemplo, asumiendo que A= $\{(1,2),(2,3)\}$, no se puede permitir que *P* sea $\{(1,2),(2,3),(1,3),(3,1)\}$ porque el resultado de la sustitución en la ecuación es:

$$\{(1,2),(2,3),(1,3),(3,1)\}=\{(1,2),(2,3)\} \cup \pi_{X,Y} (\{(1,2),(2,3),(1,3),(3,1)\} \mid x \mid \{(1,2),(2,3),(1,3),(3,1)\})$$

no es una igualdad. La unión en el lado derecho produce, por ejemplo, la tupla(3,1,2) sobre los atributos *X,Z,Y* , que después de la proyección es (3,2) una tupla que no está en el lado izquierdo.

Como un ejemplo último, un modelo que no es un punto fijo. Sea *A*={} y *P*= {(1,2)}. Entonces las reglas se hacen verdaderas. En la regla (1), no hay forma de hacer el cuerpo, *arc(X,Y)*, verdadero, así la regla es verdadera no importando que constantes son substituidas para las variables. En la regla (2), no existe valor que pueda ser sustituido para *Z* que haga que *(X,Z)* y *(Z,Y)* sean tuplas de *P*, así que el cuerpo no puede ser verdadero y la regla debe ser verdadera siempre. Se concluye que el conjunto de valores formados sólo por *camino(1,2)* es un modelo de las reglas dadas.

Sin embargo, la ecuación no se hace verdadera; su lado izquierdo es {(1,2)} y su lado derecho es {} para *A* y *P* dados. Así, *P* ={(1,2)} no es un punto fijo de las ecuaciones con respecto a EDB *A*={}.

### 5.2.2 Resolución de ecuaciones recursivas.

Se puede resolver un conjunto de ecuaciones datalog asumiendo inicialmente que todas las $P_i$ se encuentran vacías y las $R_i$ tiene el valor dado. Se aplica entonces EVAL a los valores actuales de las relaciones IDB y los valores permanentes de las relaciones EDB, para obtener nuevos valores para las relaciones IBD. Este proceso se repite hasta el punto en que ninguna de las $P_i$ cambie. Las relaciones IDB deben converger en este sentido, porque la operación EVAL es monótona, una propiedad que se debe definir más formalmente, pero que esencialmente significa que cuando se agregan más tuplas a los argumentos de la operación, el resultado no puede perder tuplas.

**Algoritmo 5.3** : Evaluación de ecuaciones de Datalog.

**Entrada:** un conjunto de reglas de datalog con predicados EDB $r_1,\ldots,r_k$ y predicados IDB $p_1,\ldots,p_m$. También una lista de relaciones $R_1,\ldots,R_k$ para servir como valores de los predicados EDB.

**Salida:** la solución punto fijo mínima para las ecuaciones datalog obtenida de esas reglas.

**Método:** se inicia calculando las ecuaciones para las reglas. Estas ecuaciones tienen variables $P_1,\ldots,P_m$ correspondientes a los predicados IDB y la ecuación para $P_i$ es $P_i$= EVAL($p_i$,$R_1,\ldots,R_k,\ldots,P_m$). Luego, cada $P_i$ se inicializa con el conjunto vacío y repetidamente se aplica EVAL para obtener nuevos valores para $P_i$. Cuando no se pueden agregar más tuplas a cualquier relación IDB, ya se tiene la salida deseada. La siguiente figura muestra un programa para más detalles.

**Ejemplo 5.5:** considérense las reglas de la figura 4.1 y la relación particular *P* para el predicado EDB de *padre*, que se muestra en la siguiente figura. En esa figura, una arista descendente de *x* a *y* indica que *x* es padre de *y*; por ejemplo, *padre(y,x)* es verdadera. Las fórmulas EVAL par los predicado *hermano*, *primo* y *pariente* o sus variables de relación equivalentes *S*, *C* y *R*, son las fórmulas dadas en el lado derecho de las ecuaciones en el

como *f* y *g*, son primos de los hijos de *d*, nombrados como *h* y *i*, así que se agrega (*f*,*h*), (*f*,*i*), (*g*,*h*) y (*g*,*i*) a *C*. Por simetría, el par primo (*d*,*c*) causa lo contrario de cada par a ser agregado en *C*, pero no se listan estos pares por la convención.

En el tercer ciclo, la regla (3) para *primo* causa (*j*,*j*) y (*k*,*k*) para ser agregado a *C*. Por ejemplo, el hecho de que *f* y *h* fueron descubiertos para ser primos en el ciclo 2, indica en el tercer ciclo que *j* es su propio primo, un resultado no intuitivo pero técnicamente correcto.

Las reglas de la (4) a (6) para *pariente* están aplicadas similarmente iniciando en el segundo ciclo. Toma hasta el ciclo 6 para que todas la tuplas de *R* sean deducidas. Por ejemplo, el hecho de que *j* es pariente de el mismo no se deduce hasta ese ciclo. Nótese que *j* está relacionado con el mismo porque *c* y *d* son hermanos y *j* es un descendiente de *c* y *j* es también descendiente de *d*.

| | S | C | R |
|---|---|---|---|
| 1 | (c,d) (d,e)<br>(f,g) (h,i)<br>(f,i) | | |
| 2 | | (f,g) (f,i) (i,i)<br>(g,h) (g,i)<br>(h,i) (j,k) | (c,d) (d,e)<br>(f,g) (h,i)<br>(f,i) |
| 3 | | (j,j) (k,k) | (d,f) (d,g) (c,h) (d,i)<br>(c,i) (e,h) (e,i) (g,j)<br>(f,k) (h,k) (i,j) |
| 4 | | | (f,h) (d,j) (g,h) (j,k)<br>(g,i) (d,k) (c,j) (i,i)<br>(c,k) (e,j) (e,k) |
| 5 | | | (f,j) (h,j) (g,k) (i,k) |
| 6 | | | (j,j) (k,k) |

**Figura 5.3 Aplicación del algoritmo 5.3**

## 5.3 Evaluación incremental de puntos fijos mínimos.

Hay que hacer notar que, en el ejemplo 5.5, cuando se están calculando nuevos valores para las $P_i$, en la figura 5.1 , se insistió en la pregunta de qué tuplas habían sido agregadas a las relaciones en el ciclo anterior y se preguntó qué nuevas tuplas había producido para su propia relación o para otra relación IDB. Esa restricción del problema es válida, porque cuando se evalúa la asignación

$$P_i := \text{EVAL}\ (p_i, R_i, \ldots, R_k, Q_1, \ldots, Q_m)$$

se da importancia solamente a las tuplas en la expresión del lado derecho que no saben todavía que están en $P_i$ .

PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

Es importante darse cuenta que cuando se ejecuta el procedimiento EVAL, para cada tupla $\mu$ que es producida se puede identificar una regla particular para $p_i$, de donde viene $\mu$ . Puede existir más de una regla que produce esta tupla se puede tomar cualquiera de esas tuplas. Además, para cada submeta de esa regla, se puede identificar una tupla de la relación para esa submeta que se usa para ayudar a producir $\mu$.

**Ejemplo 5.6** : considérense los datos del ejemplo 5.5. La tupla$(f,h)$ agregada a $C$ en el segundo ciclo viene de la regla (2) de la figura 4.1 para *primo*, y, en particular, de las tuplas $(f,c)$ en *padre*$(X,X_p)$, $(h,d)$ en *padre*$(Y,Y_p)$ y $(c,d)$ en *hermano*$(X_p,Y_p)$, no se necesitan tuplas entre la relación para hermano y (c,d), y la única razón por la que se le dé importancia sólo a dos diferentes tuplas en la relación para *padre* es porque existen dos submetas en la regla (2) con el predicado *padre*. Sólo una tupla para cada submeta contribuye a la prueba que (f,h) está en $C$.

Las nuevas tuplas producidas por cada regla se pueden encontrar si se sustituye la relación completa para todas, pero una de las submetas y sustituir sólo las tuplas increméntales, por ejemplo, las tuplas encontradas en el ciclo anterior para la submeta faltante. Las razón es que si la tupla $\mu$ no se agrega hasta el $i$ ésimo ciclo, entonces debe de existir por lo menos una submeta $S$, con una tupla $\sigma$ que $\mu$ necesita, tal que $\sigma$ no fue agregada a la relación para $S$ hasta el ciclo $i$-1. Por lo tanto, $\sigma$, es una tupla incremental en el ciclo $i$ y cuando se usan las tuplas increméntales para $S$ (y las relaciones completas para las otras submetas se generará $\mu$.

En principio, se debe hacer esta sustitución usando cada submeta en turno como la submeta con la relación incremental y, entonces, tomar la unión de las relaciones resultantes. Sin embargo, como puede que no existan tuplas increméntales para las relaciones EDB, se puede tomar la unión sobre las submetas sólo con predicados IDB, excepto en el primer ciclo. En el primer rango se debe usar las relaciones completas para todos los predicados. Sin embargo, ya que los predicados IDB tienen relaciones vacías en el ciclo 1, sólo se usan las relaciones EDB en el rango 1.

Para definir más formalmente la operación de evaluación incremental de las relaciones asociadas con reglas y predicados, sea $\tau$ una regla con las submetas ordinarias $S_1,\ldots,S_n$; se excluye de esta lista cualquier submeta con predicados construidos. Sea $R_1,\ldots,R_n$ la relación actual asociada con las submetas $S_1,\ldots,S_n$, respectivamente y sea $\Delta R_1,\ldots,\Delta R_n$ la lista de las relaciones increméntales correspondientes, el conjunto de tuplas agregado a $R_1,\ldots,R_n$ en el ciclo más reciente. Hay que recordar que REGLA-EVAL$(r,T_1,\ldots,T_n)$ es la expresión algebraica usada por el algoritmo 5.1 para computar la relación para el cuerpo de la regla $\tau$, cuando ese algoritmo usa la relación $T_i$ como la relación para la submeta $S_i$ ($T_i$ es $R_i$ en el algoritmo 5.1), entonces la relación incremental para la regla $\tau$ es la unión de las $n$ relaciones

$$\text{REGLA-EVAL}(r,R_1,\ldots,R_{i-1},\Delta R_i,R_{i+1},\ldots,R_n)$$

para $1 \leq i \leq n$. Esto es, en cada término exactamente una relación incremental está sustituida por la relación completa, formalmente se define

$$\text{REGLA-EVAL-INCR}(r,R_1,\ldots,R_n,\Delta R_1,\ldots,\Delta R_n) =$$
$$\bigcup_{1\leq i\leq n} \text{REGLA-EVAL}(r,R_1,\ldots,R_{i-1},\Delta R_i,R_{i+1},\ldots,R_n)$$

Hay que recordar que todas las reglas se asumen rectificadas, así que la unión es apropiada aquí igual que en el algoritmo 5.3.

Ahora, supóngase que se dan las relaciones $R_1,\ldots,R_k$ para los predicados EDB $r_1,\ldots,r_k$. Para los predicados $p_1,\ldots,p_m$ se dan las relaciones asociadas $P_1,\ldots,P_m$ y asociadas las relaciones increméntales $\Delta P_1,\ldots,\Delta P_m$, sea $p$ un predicado IDB. Se define

EVAL-INCR($p,R_1,\ldots,R_k,P_1,\ldots,P_m,\Delta P_1,\ldots,\Delta R_n$)

para hacer la unión de lo que REGLA-EVAL-INCR produce para cada regla para $p$. En cada aplicación de REGLA-EVAL-INCR, las relaciones increméntales para los predicados EDB es 0, así que los términos de esas submetas que son predicados EDB no tienen que aparecer en la unión para REGLA-EVAL-INCR.

**Ejemplo 5.7:** nuevamente considérense las reglas de la figura 4.1. Sea $P$, $S$, $C$ y $R$ las relaciones para *padre*, *hermano*, *primo* y *pariente*, y sea $\Delta S$, $\Delta C$ y $\Delta R$ las relaciones increméntales para los tres últimos de estos predicados que son relaciones IDB. Ya que *hermano* está definido solamente en términos de la relación EDB *padre*, se encuentra

EVAL-INCR(*hermano*,$P$)= 0

Que es REGAL-EVAL-INCR para la regla(1) es la unión sobre un conjunto vacío de submetas que tienen predicados IDB. Esta situación no es alarmante ya que como se vio en el ejemplo 5.5, $S$ tendrá todas la tuplas en el primer ciclo (la evaluación incremental inicia aplicando EVAL(*hermano*,$P$) una vez.

El predicado *primo* está definido por las reglas (2) y (3) y estas reglas cada una tiene solamente un predicado IDB: *hermano* en (2) y primo en (3). Así, para cada una de estas reglas REGLA-EVAL-INCR tiene solamente un término y la fórmula para *primo* tiene la unión de los términos para cada una de las dos reglas:

EVAL-INCR(*primo*,$P,S,C,\Delta S,\Delta C$) =

$$\pi_{X,Y}(P(X,X_P) \;|x|\; P(Y,Y_P) \;|x|\; \Delta S(X_P,Y_P)) \;\cup$$
$$\pi_{X,Y}(P(X,X_P) \;|x|\; P(Y,Y_P) \;|x|\; \Delta C(X_P,Y_P))$$

Finalmente, la fórmula de evaluación incremental para *pariente* se construye en una forma similar de las reglas de la (4) a la (6); esto es:

EVAL-INCR(*pariente*,$P,R,S,\Delta R,\Delta S$) $= \Delta S(X,Y) \cup$

$$\pi_{X,Y}(\Delta R(X,Z) \;|x|\; P(Y,Z)) \cup \pi_{X,Y}(\Delta R(Z,Y) \;|x|\; P(X,Z))$$

## 5.4 Evaluación semi ascendente.

Esta definición se usa en el siguiente mejoramiento del algoritmo 5.3. El siguiente algoritmo, tomando ventaja de las relaciones increméntales, se llama algunas veces "semi-ascendente" comparado con el algoritmo 5.3 más simple y menos eficiente que se llama "ascendente"

**Algoritmo 5.4:** Evaluación Semi Ascendente de Ecuaciones de Datalog

Entrada: una colección de reglas de datalog rectificadas, con predicados EDB $r_1,\ldots,r_k$ y predicados IDB $p_1,\ldots,p_m$. También una

lista de relaciones $R_1,\ldots,R_k$ para servir como valores de los predicados EDB.

Método: se usa EVAL una vez para obtener el cómputo de las relaciones iniciadas y, entonces, usar EVAL-INCR repetidas veces sobre relaciones increméntales IDB. El cómputo se muestra en la figura 5.4, donde para cada predicado IDB $p_i$, existe una relación $P_i$ asociada que contiene todas las tuplas y existe una relación incremental $\Delta P_i$ que tiene las tuplas agregadas en el ciclo anterior.

```
for i = 1 to m do begin
      ΔP_i = EVAL(P_i, R_1,...,R_k, 0,...,0);
      P_i = ΔP_i
end;
repeat
      for i = 1 to m do
            ΔQ_i=ΔP_i /* almacena los ΔP anteriores */
      for i = 1 to m do begin
            ΔP_i =EVAL-INCR(p_i, R_1,...,R_k, P_1,...,P_m, ΔQ_1,...,ΔQ_m);
            ΔP_i =ΔP_i - P_i /* remueve las tuplas nuevas
                              que aparecieron anteriormente */
      end;
      for i = 1 to m do
            P_i =P_i U ΔP_i
Until ΔP_i = 0 para cada y
```

**Figura 5.4 Programa de evaluación semi-ascendente.**

**Ejemplo 5.8:** en el ejemplo 5.7, en el primer ciclo, que es el ciclo inicial para el *for* en la figura 5.4. se usa la operación EVAL. Y como se vio en el ejemplo 5.5, sólo la relación $S$ para hermano, obtiene alguna tupla en este ciclo, porque este predicado tiene una regla sin predicados IDB en el cuerpo. Así, en el segundo ciclo, $S$ y $\Delta S$ son ambos la relación completa para *hermano*, mientras que todas las otras IDB y las relaciones increméntales están vacías.

En el segundo ciclo, por ejemplo, la primera vez en el ciclo del repeat de la figura 5.4, $\Delta S$ se hace igual a 0, ya que esto es lo que EVAL-INCR retorna, como se discutió en el ejemplo 5.7. Los términos de las reglas (2) y (4) contribuyen con algunas tuplas para $\Delta C$ y $\Delta R$, respectivamente y estas tuplas entonces encuentran su forma en $C$ y $R$ al final del ciclo del repeat. Esto es, en el ciclo dos, se computa:

$$C = \Delta C = \pi_{X,Y}(P(X,X_p) \;|x|\; P(Y,Y_p) \;|x|\; \Delta S(X_p,Y_p))$$
$$R = \Delta R = \Delta S$$

En el tercer ciclo, ya que $\Delta S$ está vacía, las reglas (2) y (4) no pueden producir más tuplas, pero como $\Delta C$ y $\Delta R$ ahora tiene algunas tuplas, las reglas (3), (5) y (6) si pueden. Entonces, se computa:

$$\Delta C = \pi_{X,Y}(P(X,X_1) \;|x|\; P(Y,Y_p) \;|x|\; \Delta C(X_p,Y_p))$$

$$/R = \pi_{X,Y}(/R(X,Z) \ |x| \ P(Y,Z)) \ U \ \pi_{X,Y}(/R(Z,Y) \ |x| \ P(X,Z))$$

Los valores de $/C$ y $/R$ están acumulados en $C$ y en $R$ respectivamente y si ambos no están vacíos, se repite otro ciclo de la misma manera.

El algoritmo 5.4 computa correctamente los puntos fijos mínimos de sus reglas dadas y dadas las relaciones EDB.

Los algoritmos 5.3 y 5.4 computan el mismo conjunto de tuplas para cada una de las relaciones IDB en cada ciclo. Ya que el algoritmo 5.3 se mostró para computar los puntos fijos mínimos, entonces se concluye que el algoritmo 5.4 también lo hace. La hipótesis inductiva actual que se necesita es que una tupla agregada a alguna relación IDB $P$ en el ciclo $j$ por el algoritmo 5.3 no ha sido colocada en esa relación en cualquier otro ciclo anterior, será colocada en $P$ y $/P$ en el ciclo $j$ del algoritmo 5.4. Las bases, el ciclo 1, es inmediato, ya que las mismas fórmulas dadas por EVAL son usadas por ambos algoritmos.

Para la inducción, sólo se debe notar que si una tupla $\mu$ se agrega a alguna relación IDB $P$ en el ciclo $i$, y $\mu$ no estaba previamente en $P$, entonces, debe existir alguna regla $r$ para el predicado $p$ ( el predicado correspondiente a la relación $P$) y las tuplas en las relaciones para todas las submetas de $r$ tal que:

1. las tuplas para las submetas, juntas, producen $\mu$, y,

2. al menos una de estas tuplas, digamos $\tau$, fue agregada a su relación, digamos $T$, en el ciclo $i-1$.

Por la hipótesis inductiva con $j = i-1$ y la observación (2) arriba, $\tau$ está en $/T$ cuando se inicia el ciclo $i$ del algoritmo 5.4. Por lo tanto, el término de EVAL-INCR que usa $/T$ producirá $\mu$ ya que ese término usa las relaciones completas para las submetas, otra que la que suministra $\tau$ y $\tau$ será suministrada por $/T$.

**5.5 Negaciones en el cuerpo de las reglas.**

Existen situaciones frecuentes donde sería más fácil usar la negación de un predicado para ayudar a expresar una relación con reglas de la lógica. Técnicamente, reglas con submetas negadas no son cláusulas de Horn, pero se puede ver que muchas de las ideas desarrolladas se aplican a este extenso tipo de reglas. Generalmente, el significado intuitivo de una regla con una o más submetas negadas es lo que se debe complementar en las relaciones para las submetas negadas y luego computar la relación de la regla exactamente como se hizo en el primer algoritmo.

Desafortunadamente, el complemento de una relación no es un término bien definido. Se debe especificar la relación o dominio de los posibles valores respecto del complemento que se tome. Por eso es que el Álgebra relacional usa un operador diferenciador, pero no un operador de complemento . Pero aunque se especifique el universo de las posibles tuplas respecto de dónde se compute el complemento de una relación, se puede decir que este complemento será una relación normalmente infinita. No se puede, por lo tanto, aplicar operaciones como las de selección y unión al complemento y no se puede ejecutar el algoritmo mencionado inicialmente, sobre una regla con negación de una manera directa.

En el próximo ejemplo se observa qué pasa cuando las cosas trabajan correctamente y entonces se muestra donde los problemas surgen cuando las variables aparecen sólo en submetas negadas. Luego se examina otro ejemplo donde algunas submetas están negadas: no es necesario un punto fijo más pequeño para un programa de la lógica. Además, si no se tiene un mecanismo para probar los datos negados, el punto de vista de la prueba teórica no ayuda en cosa alguna y se debe forzar a seleccionar uno de los modelos mínimos como el significado de un programa de lógica.

**Ejemplo 5.9** : supóngase que se quiere definir "primo verdadero" a los individuos que se encuentran relacionados que el predicado de *primo* pero no al mismo tiempo relacionado por la relación de *hermano*. Se debería escribir

primoVerdadero(X,Y):- primo(X,Y) & -hermano(X,Y)

Esta regla se parece mucho a la aplicación de diferencia del Álgebra relacional y realmente se puede computar $T = C - S$, donde $T$ es la relación para *primoVerdadero* y $C$ y $S$ son las relaciones para *primo* y *hermano*, computados anteriormente en otro ejemplo.

La fórmula $T = C - S$ es fácil de ver para dar la misma relación como:
$C(X,Y)$ $|X|$ $-S(X,Y)$

donde $-S$ es el complemento de $S$ con respecto a algún universo $U$ que incluye por lo menos las tuplas de $C$. Por ejemplo, $U$ puede ser el conjunto de individuos que aparecen en una o más tuplas de la relación de *padre*. Entonces, $-S$ podría ser $UxU-S$ y seguramente $C$ es un subconjunto de $U$ x $U$.

Desafortunadamente, no todos los usos de la negación son tan directos como el del ejemplo anterior. Existen otros problemas al del significado de las reglas con negación y, entonces, desarrollar un conjunto de restricciones sobre el uso de la negación que permiten a las reglas, con esta forma limitada de negación, es un significado sensible. El primer problema que se encuentra es cuando las variables aparecen sólo en submetas negadas.

**Ejemplo 5.10**: considérese la siguiente regla:

soltero (X) := hombre(X) & ¬casado(X,Y).

Aquí se supone que *hombre* es una relación EDB y su significado es obvio y *CASADO(X,Y)* es una relación EDB con el significado de que $X$ es el esposo de $Y$.

Una posible interpretación de la regla anterior es que $X$ es un soltero si el es un hombre y no existe una $Y$ tal que $Y$ esté casada con $X$. Sin embargo, si se computa la relación para esta regla uniendo la relación *hombre(X)* con el complemento de *casado*, esto es, con el conjunto de pares *(X,Y)* tal que $X$ no está casado con $Y$, se puede obtener el conjunto de pares *(X,Y)* tal que $X$ es hombre y $Y$ no está casada con $X$. Si se proyecta este conjunto sobre $X$, se encuentra que los solteros son hombres que no están casados absolutamente con nadie en el universo; esto quiere decir que existe alguna $Y$ tal que $Y$ no está casada con $X$.

Para evitar esta aparente divergencia entre lo que intuitivamente se espera que la regla debería significar y la respuesta que se obtiene si se interpreta la relación en la forma obvia(el complemento de la relación), se debe prohibir el uso de una variable en una submeta negada si esa variable no

aparece también en otra submeta y que esa submeta ni está negada ni es un predicado construido. Esta restricción no es severa, ya que siempre se puede reescribir una regla de tal manera que esas variables no aparezcan. Por ejemplo, para hacer que los atributos de, las dos relaciones involucradas en la regla anterior, sean los mismos, debe hacerse una desproyección de *Y* de *casado*; esto es, las reglas se deben reescribir como:

```
esposo(X) :- casado(X,Y).
soltero(X) :- hombre(X) & ¬esposo(X).
```

Estas reglas pueden tener sus significados expresados por:

$$esposo(X) = \pi_X(casado(X,Y))$$
$$soltero(X) = hombre(X) - esposo(X)$$

o solamente:

$$soltero(X) = hombre(X) - \pi_X(casado(X,Y))$$

Mientras se prohiba el uso de variables que aparecen solamente en submetas negadas, se debe tener claro que el conjunto de variables en una submeta negada que encaja exactamente con las variables de las submetas no negadas, no es esencial. El próximo ejemplo da una idea de qué se puede hacer en casos donde hay muy pocas variables en una submeta negada.

**Ejemplo 5.11**: considérese:

```
puedeComprar(X,Y) :- parecido(X,Y) & ¬quebrado(x)
```

Aquí, *parecido* y *quebrado* se presumen como relaciones EDB. La intención de esta regla, evidentemente, es que *X* pueda comprar *Y* si *X* es parecido a *Y* y *X* no está quebrado. Hay que recordar que la relación para esta regla es una unión involucrando el complemento de *quebrado*, que debería llamar *noQuebrado*. La siguiente regla puede ser expresado por la ecuación equivalente del álgebra relacional:

$$puedeComprar(X,Y) = parecido(X,Y) \ |X| \ noQuebrado(X)$$

El hecho de que *noQuebrado* puede ser infinito no previene el procesamiento del lado derecho de la ecuación anterior, porque se puede iniciar con todas las tuplas de *parecido(X,Y)* y luego chequear que cada una tiene un componente X que es un miembro de *noQuebrado*, o, equivalente, no es miembro de *quebrado*.

Como se hizo en los dos ejemplos anteriores, la ecuación anterior puede ser expresada con un conjunto finito de relaciones, si se rellenan las tuplas de *quebrado* con todos los objetos posibles que se pueden parecer. Pero no existe una forma de decir "todos los objetos" en el álgebra relacional ya que es un conjunto infinito.

Hay que darse cuenta que no se necesitan todos los pares (X,Z) tal que X está quebrado y Z sea cualquier cosa, ya que todos , pero un número finito de Z, aparecerá como un segundo componente de una tupla *parecido* y por lo tanto, no es posible que esté en la relación *puedeComprar*. El conjunto posible para Z está expresado en el Álgebra relacional como $\pi_2(parecido)$ o equivalentemente, $\pi_Y(parecido(X,Y))$. Entonces, se debe expresar *puedeComprar*, en álgebra relacional como:

$$puedeComprar(X,Y) = parecido(X,Y) - (quebrado(X) \mid X \mid \pi_Y(parecido(X,Y)))$$

Finalmente, se puede derivar de la expresión anterior una forma de expresar *puedeComprar* con reglas donde la única literal negada aparece en una regla con una literal positiva que tiene exactamente el mismo conjunto de variables. Tales reglas pueden ser interpretadas naturalmente. La idea general es usar una regla para obtener la proyección sobre el conjunto de valores necesarios, $\pi_2(parecido)$ en este caso, usar entonces otra regla para rellenar las tuplas en la relación negada. Las reglas para este caso quedan como:

```
parecido2(Y) :- parecido(X,Y).
parQuebrado(X,Y) :- quebrado(X) & parecido2(Y)
puedeComprar(X,Y):- parecido(X,Y) & ¬parQuebrado(X,Y).
```

### 5.5.1 No unicidad de puntos fijos mínimos.

Ajustando el conjunto de atributos en las diferencias de las relaciones es importante, pero no resuelve todos los problemas potenciales de las submetas negadas. Si $S_1$ y $S_2$ son dos soluciones para un programa de lógica, respecto de un conjunto dado de relaciones EDB, se dice que $S_1 < S_2$ si $S_1 \leq S_2$ y $S_1 \neq S_2$. Hay que recordar que se dice que el punto fijo $S_1$ es mínimo si no existe un punto fijo tal que $S < S_1$. También se dice que $S_1$ es el punto fijo más pequeño si $S_1 \leq S$ para todos los puntos fijos $S$. Cuando se permiten reglas con negación, no debería existir un punto fijo más pequeño, pero sí algunos puntos fijos mínimos. Si no existe un único punto fijo más pequeño, ¿qué es lo que significa un programa de lógica?

**Ejemplo 5.12:** considérese las siguientes reglas:

```
(1) p(x) :- r(x) & ¬q(x).
(2) q(x) :- r(x) & ¬p(x).
```

Sean $P$, $Q$ y $R$ las relaciones IDB para los predicados $p$ y $q$ y el predicado EDB $r$, respectivamente. Supóngase que $R$ consiste de una tupla simple; $R = \{1\}$. Sea $S_1$ la solución $P = 0$ y $Q = \{1\}$; sea $S_2$ igual a $P = \{1\}$ y $Q = 0$. Ambas $S_1$ y $S_2$ son soluciones para las ecuaciones $P = R - Q$ y $Q = R - P$.

Hay que notar que las reglas (1) y (2) son lógicamente equivalentes , pero las dos ecuaciones no lo son; ciertos conjuntos de $P$, $Q$ y $R$ satisfacen una pero no la otra.

Obsérvese que $S_1 \leq S_2$ es falsa, por los valores respectivos de $Q$, y $S_2 \leq S_1$ es falsa por los de $p$. Además no existe una solución $S$ tal que $S < S_1$ o $S < S_2$. La razón es que alguna $S$ puede asignar 0 a $P$ y a $Q$ . Pero entonces $P = R - Q$ no cuadraría.

Se puede concluir, entonces, que $S_1$ y $S_2$ son puntos fijos y que ambos son mínimos. Así, el conjunto de reglas anteriores no tiene un punto fijo más pequeño, ya que si existiera ese punto fijo $S$, se tendría $S < S_1$ y $S < S_2$.

### 5.5.2 Negación estratificada.

Para ayudar con el problema de los puntos fijos mínimos, se debe permitir solamente la negación estratificada. Formalmente, las reglas están estratificadas si cuando existe una regla con un predicado $p$ de cabeza y una submeta negada con predicado $q$, no existe un camino en el grafo de dependencias de $p$ a $q$. La restricción de reglas para permitir sólo negaciones estratificadas no garantiza un punto fijo más pequeño. Sin embargo, permite una selección racional entre los puntos fijos mínimos, proporcionando uno que venga a ser generalmente aceptado como "el significado" de un programa de lógica con negación estratificada.

Hay que mencionar que el grafo de dependencias debe construirse de igual manera que si el signo de negación no existiera.

**Ejemplo 5.13:** considérese las reglas estratificadas.

```
(1) p(X) :- r(X).
(2) p(X) :- p(X).
(3) q(X) :- s(X) & ¬p(X)
```

Este conjunto de reglas está estratificado ya que la única ocurrencia de una submeta negada, $\neg p(X)$ en la regla (3), tiene un predicado cabeza, $q$, de donde no existe un camino a $p$ en el grafo de dependencias, esto es, aunque $q$ depende de $p$, $p$ no depende de $q$.

Sean $r$ y $s$ relaciones EDB y sus relaciones correspondientes $R$ y $S$ y sean $p$ y $q$ relaciones IDB y sus relaciones correspondientes $P$ y $Q$. Supóngase que $R=\{1\}$ y $S=\{1,2\}$. Entonces, una solución es $S_1$ dada por $P=\{1\}$ y $Q=\{2\}$, otra puede ser $S_2$ dada por $P=\{1,2\}$ y $Q=0$, esto es que $S_1$ y $S_2$ son soluciones para las ecuaciones $P = P \cup R$ y $Q = S - P$.

Se puede chequear que $S_1$ y $S_2$ son mínimos. Así, no puede existir un punto fijo más pequeño para las reglas de este ejemplo. Hay que notar también que $S_1$ es más natural ya que cada una de sus tuplas se puede obtener haciendo sustituciones de datos conocidos en los cuerpos de las reglas y deduciendo el valor que aparece en la cabeza.

### 5.5.3 Encontrando estratificaciones.

Ya que no todos los programas de lógica con negaciones están estratificados, es usual tener un algoritmo para probar la estratificación. Resulta fácil hacer esta prueba y la misma proporciona la estratificación de las reglas, esto lo hace agrupando los predicados, que son los grupos de predicados mas largos tales que:

1. si un predicado $p$ está en la cabeza de una regla con una submeta que es una $q$ negada, entonces $q$ está en una clase menos que $p$;

2. si el predicado $p$ está en la cabeza de una regla con una submeta que es una $q$ no negada, entonces, la clase de $p$ es por lo menos tan alta como la clase de $q$.

La clase da un orden en el que las relaciones para los predicados IDB pueden ser procesadas. El uso propio de este orden, es que siguiéndolo, se puede tratar cualquier submeta negada como si fuera una relación EDB.

**Algoritmo 5.5:** Para probar y encontrar una estratificación.

**Entrada:** un conjunto de reglas de datalog, posiblemente con algunas submetas negadas.

**Salida:** una decisión ya que las reglas están estratificadas. Si es así, también se produce una estratificación.

**Método:** iniciar con cada predicado asignándolos a la clase 1. Repetidamente examinar las reglas. Si una regla con el predicado $p$ de cabeza tiene una submenta negada con predicado $q$, entonces $p$ y $q$ se asignan a la clase $i$ y $j$ respectivamente. Si $i \leq j$ reasignar $p$ a la clase $j+1$. Además, si una regla con cabeza $p$ tiene una submeta no negada con predicado $q$ de estado $j$, y $i < j$ , reasignar $p$ a la clase $j$. Estas leyes se formalizan en la figura 5.7.

Si se tiene una condición donde no se puede cambiar la clase por el algoritmo de la figura 5.5, entonces, las reglas están estratificadas y la clase actual forma la salida del algoritmo. Si alguna vez se tiene una condición donde algún predicado es asignado a una clase que es más grande que el número total de predicados, entonces, las reglas no están estratificadas, entonces, se detiene el proceso y se produce como salida "no".

```
PARA cada predicado p  HAGA
 clase[p] := 1;
REPITA
   PARA cada regla r con predicado p  en la cabeza  HAGA
     INICIO
        PARA cada submeta negada de r con predicado q
          HAGA  clase[p] := MAX(clase[p], 1+clase[q]);
        PARA cada submeta no negada de r con predicado q
          HAGA  clase[p] := MAX(clase[p], clase[q])
     FIN
HASTA que no existan cambios en ninguna clase  ó
      alguna clase exceda el numero de predicados
```

**Figura 5.5 Proceso de estratificación.**

**Ejemplo 5.14:** si no existen submetas negadas, entonces, el algoritmo 5.5 termina inmediatamente con todos los predicados en la clase 1, que es correcto.

Para un ejemplo menos trivial, considérense las reglas del ejemplo 5.12. Inicialmente, $p$ , $q$ y $r$ cada una tiene clase 1. La regla (1) obliga a incrementar la clase de $p$ a 2, y, entonces, la regla 2 obliga a incrementar la clase de $q$ a 3. La primera regla requiere que la clase de $p$ sea 4. Se tiene ahora un estado más grande que el número de predicados y así se concluye que las reglas no están estratificadas. Y esa conclusión es correcta. Por ejemplo, $q$ aparece como una submeta negada en la primera regla, que tiene como cabeza el predicado $p$, $q$ aún depende de $p$ .

Para otro ejemplo, considérense las reglas del ejemplo 5.13. Iniciando con los cuatro predicados en la clase 1, la regla (3) obliga a incrementar *q* a la clase 2. Sin embargo, no existen más ajustes que sea necesario hacer. Se concluye que *p*, *r* y *s* están en la clase 1. Esto tiene sentido, porque *r* y *s* son relaciones EDB, mientras que *p* es una relación IDB que puede ser procesada de éstas por el algoritmo 5.3 o 5.4, sin ningún uso de negación. Después de haber procesado *p*, se puede pretender que la negación de *p* es una relación EDB. Ya que *q* no es recursiva, se puede procesar la relación para *q* por el algoritmo 5.2.

Se puede decir que un programa de lógica que tiene una estratificación, entonces está estratificado. No se debe tomar con calma la similitud de los términos estratificados y tiene una estratificación con la idea que probará fácilmente su equivalencia. De hecho son equivalentes, pero su prueba requiere de algún trabajo. Hay que recordar que un programa está estratificado cuando existe una submeta negada con predicado *q* en el cuerpo de una regla para el predicado *p*, y no existe un camino en el grafo de dependencias de *p* a *q*. Es fácil ver de la definición de una estratificación que la clase de predicados entre cualquier camino en el grafo de dependencias nunca puede decrementar, porque esos caminos van de la submeta a la cabeza y la clase de un predicado cabeza nunca es menor que la clase de una de sus submetas.

Supóngase que un programa tiene una estratificación, pero no fue estratificada. Entonces, podría existir un camino en el grafo de dependencias a alguna *q* de alguna *p*, de tal manera que *q* negada era una submeta de una regla *r* para *p*. La existencia del camino indica que la clase de *q* es, por lo menos, tan grande como la clase de *p*, la regla *r* aún requiere que la clase de *q* sea menor que la de *p*.

Se puede decir también que si un programa de lógica está estratificado, entonces, el algoritmo 5.5 se detiene sin producir una clase mayor que *n*, el número de predicados del programa.

Cada vez se incrementa la clase de algún predicado *p* por algún predicado *q* en el algoritmo de la figura 5.5, eso debe ser que *q* es una submeta (negada o no) de una regla para *p*. Si se incrementa *clase[p]* a *i* y *q* no está negada, entonces, se escribe $q \rightarrow^{i} p$; si *q* está negada, escribir $q \Rightarrow^{i} p$. Por ejemplo, la secuencia de clases cambiada para las reglas no estratificadas del ejemplo 5.12 es $q \Rightarrow^{2} p \Rightarrow^{3} q \Rightarrow^{4} p$.

Por razones técnicas es conveniente agregar un nuevo símbolo *inicio* que se asume no será un predicado. Entonces, se puede escribir $inicio \Rightarrow^{1} p$ para todos los predicados *p*.

Esto es una fácil inducción en el número de veces que el algoritmo 5.5 cambia una clase que si se inicializa la clase de un predicado *p* con *i*, entonces, existe una cadena de los pasos $\rightarrow$ y $\Rightarrow$ de *inicio* a *p* que incluye por lo menos *i* $\Rightarrow$ pasos. El punto clave en esta prueba es que si el último paso para el que el algoritmo 5.5 hace la clase de *p* tenga *i*, es $q \Rightarrow^{i} p$, entonces, existe una cadena con por lo menos *i-1* pasos a *q* y uno más hace por lo menos los *i*$\Rightarrow$ a *p*. Si el paso por el que el algoritmo 5.5 hace la clase de *p* alcanzar *i* es $q \rightarrow^{i} p$, entonces, ya existe una cadena que incluye por lo menos *i*$\Rightarrow$ a *q* y esta cadena se puede extender a *p*.

Ahora debe notarse que si de la clase de algún predicado alacanza *n+1*, entonces, existe una cadena con, por lo menos, *n+1* $\Rightarrow$. Así, algún predicado,

dígase $p$, aparece dos veces como la cabeza de una =>. Así, una parte de la cadena es :

$$q_1 =>^i p .. q_2 =>^j p$$

donde $i < j$. Además, se debe observar que cada porción de la cadena es un camino en el grafo de dependencias; en particular, existe un camino de $p$ a $q_2$ en el grafo de dependencias.

El hecho que $q_2 =>^j p$ es un paso, implica que existe una regla con cabeza $p$ y una submeta negada $q_2$. Así, existe un camino, en el grafo de dependencias de la cabeza $p$, de alguna regla, a la submeta negada $q_2$, de esa regla, contradiciendo lo asumido de que el programa lógico está estratificado. Se concluye entonces, que si el programa está estratificado, ninguna clase producida por el algoritmo 5.5 excederá $n$, y, por lo tanto, el algoritmo 5.5, eventualmente, debe ser detenido y contestar que "si".

Entonces, se puede decir que el algoritmo 5.5 determina correctamente cuándo un programa de datalog con negaciones está estratificado.

### 5.5.4 Reglas estratificadas seguras.

En el orden de que el significado sensible de las reglas puede ser definido, se necesita más que estratificación; se necesita seguridad. Hay que recordar que se definió que una regla es segura si todas sus variables están limitadas, ya sea por argumentos de una submeta ordinaria no negada o por ser igual a una constante o a una variable limitada, tal vez a través de un cadena de igualdades. Cuando se tienen submetas negadas, la definición de seguridad no cambia. No se permite el uso de submetas negadas para ayudar a probar que las variables están limitadas.

**Ejemplo 5.15:** las reglas de los ejemplos 5.11, 5.12 y 5.13 son todas seguras. La regla del ejemplo 5.10 no es segura ya que $Y$ aparece en una submeta negada pero en una submeta no negada, y, por lo tanto, no puede ser limitada. Sin embargo, como se vio en el ejemplo se puede convertir esa regla en un par de reglas seguras que intuitivamente significan lo mismo.

Cuando las reglas son seguras y están estratificadas, existe una opción natural de entre los posibles puntos fijos que se considerará como el significado de las reglas. Se procesa cada clase en orden, iniciando por el más pequeño. Asúmase que se está trabajando con un predicado $p$ de clase $i$. Si una regla para $p$ tiene una submeta con un predicado $q$ de clase menor que $i$, se puede obtener la relación de $q$, ya que esa relación también es una relación EDB o ha sido computada cuando se procesó en una clase previa. Claro está que no hay submetas que puedan ser de clase menor que $i$ , si se tiene una estratificación válida. Además, si la submeta está negada, entonces, la clase de $q$ debe ser estrictamente menor que $i$.

Como una consecuencia de estas propiedades de una estratificación, se puede visualizar el conjunto de reglas para los predicados de clase $i$ como una definición recursiva de las relaciones exactamente para los predicados de clase $i$, en términos de relaciones para las relaciones EDB y todas la relaciones IDB de menor clase. Como las ecuaciones para los predicados IDB de clase $i$ no tienen submetas negadas de clase $i$, se deben aplicar los algoritmos 5.3 o 5.4 para resolverlas.

Lo único técnico, concierne a cómo crear la relación para una submeta negada $\neg q(X_1,...,X_n)$ de una regla $r$, así que se debe pretender que es una relación finita que viene a ser una submeta negada. Se define a *DOM* como la unión de los símbolos que aparecen en la relación EDB y en las mismas reglas. Como ya se mencionó, en reglas seguras, símbolos que no están en la relación EDB o en las reglas, pueden aparecer en una sustitución que pueden hacer el cuerpo de una regla verdadera. Por lo tanto, no se pierde nada por la restricción de la relación para la submeta negada a consistir únicamente las tuplas cuyos valores están seleccionados de *DOM*.

Así sea $Q$ la relación ya procesada para $q$ ( o dada, si $q$ es un predicado EDB). Sea la relación $\neg Q$ para la submeta $\neg q(X_1,...,Xn)$

*DOM X ...X DOM (n veces) - Q*

Si se hace una sustitución análoga para cada submeta negada en las reglas para la clase $i$, y, luego, aplicar el algoritmo 5.3 para procesar los puntos fijos mínimos para los predicados IDB de esa clase, $i$, se debe obtener el mismo resultado como si se hubiera manejado para obtener la relación infinita de todas las tuplas que no están en $Q$ en lugar de $\neg Q$. La razón es que se puede probar en una inducción fácil sobre el número de ciclos del algoritmo 5.3, que cada tupla que se agrega a una relación IDB de clase $i$ que consiste de tuplas cuyos componentes están todos en *DOM*. La prueba depende únicamente de la observación de que, como las reglas son seguras, cada variable que aparece en una submeta negada aparece también en una submeta no negada. Dado esto, Se puede invocar las hipótesis inductiva para mostrar que cada ciclo de la relación para cada regla será un subconjunto de *DOM x...x DOM*. Según lo anterior, se puede escribir el siguiente algoritmo para procesar el punto fijo perfecto de un programa de datalog con una negación estratificada segura.

**Algoritmo 5.6:** Evaluación de relaciones seguras para programas estratificados de datalog.

**Entrada:** un programa de datalog cuyas reglas estén seguras, rectificadas y estratificadas. También relaciones para todos los predicados EDB de un programa.

**Salida:** relaciones para todos los predicados IDB, formado un punto fijo mínimo de un programa de datalog.

**Método:** primero computar la estratificación para el programa aplicando el algoritmo 5.5. Procesar *DOM* con la proyección de todas la relaciones EDB dentro de cada uno de sus componentes y entonces tomando la unión de esas proyecciones y el conjunto de constantes que aparecen en las reglas.

Entonces, para cada clase $i$ en turno, hacer los siguientes pasos: cuando se alcance la clase $i$ ya se han procesado las relaciones para los predicados IDB en una clase menor y, por supuesto, ya se han dado las relaciones para los predicados EDB. Así, en particular, si una regla en la clase $i$ tiene una submeta negada, la relación para esa submeta es conocida.

## 6. ÁLGEBRA RELACIONAL Y LÓGICA.

Cada expresión de álgebra relacional produce la misma relación (como una función de sus argumentos) como una expresión de Álgebra relacional cuyas únicas selecciones son de la forma $\sigma_{X\Theta Y}$, donde $X$ y $Y$ son atributos o constantes y $\Theta$ es un operador de comparación aritmético.

Se llamarán selecciones complejas a las selecciones cuyas fórmulas involucran AND, OR y NOT y selecciones simples a las que no, se asumirá que las fórmulas no tienen ocurrencias del operador NOT.

Supóngase que se tiene una expresión $\sigma_F(E)$ donde $F$ es una selección compleja que no incluye operadores NOT. Se asume que cualquier selección compleja en $E$ ya ha sido reemplazada, así que cualquier selección que se encuentre es simple. Se muestra por inducción sobre el número de operadores AND y OR en $F$ que se puede reemplazar $\sigma_F(E)$ por una expresión cuyas selecciones todas son simples. Las bases, cero operadores lógicos, $\sigma_F(E)$ cumplirá.

Para la inducción, se debe suponer que se puede escribir $F$ como $F_1 \Lambda F_2$. Entonces, se puede escribir $\sigma_F(E) = \sigma_{F2}(\sigma_{F1}(E))$. $F_1$ y $F_2$ cada uno tiene menos operadores AND y OR que $F$, así que por la hipótesis inductiva, existe una expresión $E_1$ equivalente a $\sigma_{F1}(E)$ únicamente con selecciones simples. También por la hipótesis inductiva, existe una expresión usando solamente selecciones simples que es equivalente a $\sigma_{F2}(E_1)$. Esta expresión es equivalente a $\sigma_F(E)$.

Si el operador más externo de $F$ no es $\Lambda$, entonces, debe ser V; por ejemplo, $F$ es de la forma $F1$ V $F_2$. Entonces, se puede escribir $\sigma_F(E)=\sigma_{F1}(E)$ $U$ $\sigma_{F2}(E)$. La hipótesis inductiva indica que las expresiones $E_1$ y $E_2$, libres de selecciones complejas y equivalentes a $\sigma_{F1}(E)$ y $\sigma_{F2}(E)$, respectivamente, existen. Entonces, $E_1$ $U$ $E_2$ es una expresión equivalente a $\sigma_F(E)$ teniendo únicamente selecciones simples.

**Ejemplo 6.1:** considérese la expresión

$$E = \sigma_{\neg(\$1=\$2 \Lambda (\$1<\$3 \text{ V } \$2\leq\$3))}(R)$$

Se usan las leyes de De Morgan dos veces para reemplazar la condición de selección por

$$\neg(\$1=\$2) \text{ V } \neg(\$1<\$3 \text{ V } \$2\leq\$3)$$

y luego por

$$\neg(\$1=\$2) \text{ V}( \neg(\$1<\$3) \Lambda \neg(\$2\leq\$3))$$

ahora se absorben las negaciones en las comparaciones, quedando

$$E = \sigma_{(\$1\neq \$2 \text{ V } (\$1\geq\$3 \Lambda \$2>\$3))}(R)$$

ahora como el operador más externo es V, se puede escribir:

$$E = \sigma_{\$1\neq\$2}(R) \; U \; \sigma_{\$1\geq\$3 \Lambda \$2>\$3}(R)$$

El primer argumento de la unión es simple, pero, el segundo, requiere una transformación para el operador Λ, quedando:

$$E = \sigma_{\$1 \neq \$2}(R) \ U \ \sigma_{\$1 \geq \$3}(\sigma_{\$2 > \$3}(R))$$

**Teorema 6.1:** cualquier función que se puede expresar en Álgebra relacional se puede expresar en un programa no recursivo de datalog.

Lo anterior es una fácil inducción sobre el tamaño de la expresión algebraica. Formalmente, se muestra que si una expresión tiene $i$ ocurrencias de operadores, entonces, existe un programa de datalog no recursivo que produce, así como la relación para uno de sus predicados, el valor de la expresión. Las bases son $i = 0$, que es un único operando. Si este operando es una relación dada $R$, entonces, $R$ es una relación EDB y así disponible sin la necesidad de alguna regla. La otra posibilidad es que el operando sea un conjunto de tuplas. Entonces, se debe asignar un nombre, dígase $P$, para este conjunto y para cada una de las tuplas en el conjunto dígase $a_1 a_2 \ldots a_n$, no se tiene el cuerpo de una regla de la forma:

$$p(a_1 a_2 \ldots a_n)$$

Para la inducción, considérese una expresión cuyo operador más externo es uno de unión, diferencia, selección, proyección y producto. Hay que recordar que todos los otros operadores que se introdujeron, tales como la intersección y otros se pueden expresar en términos de estos cinco operadores básicos.

Caso 1: la expresión es $E = E_1 \ U \ E_2$. Entonces, por la hipótesis inductiva, existen los predicados $e_1$ y $e_2$ definidos por reglas de datalog no recursivas cuyas relaciones son las mismas que las relaciones definidas por las expresiones $E_1$ y $E_2$. Si estas relaciones tienen grado $n$, sus grados deben ser los mismos o el operador de unión no se puede aplicar. Entonces, para la expresión $E$ se tienen las reglas:

$$e(X_1, \ldots, X_n) \ :- \ e_1(X_1, \ldots, X_n).$$
$$e(X_1, \ldots, X_n) \ :- \ e_2(X_1, \ldots, X_n).$$

así las tuplas en la relación para e son exactamente esas tuplas que están en la relación para $e_1$ o en la relación para $e_2$ o en ambas.

Caso 2: $E = E_1 - E_2$. Se asume, como en el caso 1, que las relaciones para $E_1$ y $E_2$ son cada una de grado $n$ y que existen predicados $e_1$ y $e_2$ cuyas reglas definen sus relaciones que son las mismas relaciones para $E_1$ y $E_2$, respectivamente. Entonces se usa la regla :

$$e(X_1, \ldots, X_n) \ :- \ e_1(X_1, \ldots, X_n) \ \& \ \neg e_2(X_1, \ldots, X_n).$$

para definir un predicado e cuya relación es la misma que la relación para $E$. Se puede fácilmente revisar que el algoritmo 5.6, que se debe usar porque existe una negación, asigna la relación apropiada a e.

Caso 3: $E = \pi_{i1,\ldots ik}(E_1)$. Tenga la relación de $E_1$ grado $n$, y, sea $e_1$ un predicado cuyas reglas producen la relación para $E_1$. Entonces la regla para e, el predicado correspondiente a la expresión e, es:

$e(X_{i1},\ldots,X_{ik})$ :- $e_1(X_1,\ldots,X_n)$

Caso 4 : $E = E_1$ x $E2$. Tengan $E_1$ y $E_2$ los predicados $e_1$ y $e_2$ cuyas reglas definen sus relaciones y asumir que sus relaciones son de grado $n$ y $m$, respectivamente. Entonces se define $e$, el predicado para $E$ , por:

$e(X_1,\ldots,X_{n+m})$ :- $e_1(X_1,\ldots,X_n)$ & $e_2(X_{n+1},\ldots,X_{n+m})$ .

Caso 5: $E = \sigma_F(E_1)$. Se asume que $F$ es una selección simple, dígase $\$i\ \Theta\ \$j$; el caso se refiere a uno de los operandos de $F$, es una constante, es similar. Sea $e_1$ un predicado cuya relación es la misma que la relación para $E_1$ y se supone que $e_1$ tiene grado $n$, entonces, la regla para $e$ es:

$e(X_1,\ldots,X_n)$ :- $e_1(X_1,\ldots,X_n)$ & $X_i\ \Theta\ X_j$.

**Ejemplo 6.2:** considérese la expresión algebraica:

$puedeComprar(X,Y) = parece(X,Y) - (roto\ x\ \pi_Y(parece\ (X,Y)))$

desarrollada en el ejemplo 5.11. el operador más externo es -, con el operando izquierdo $parece(X,Y)$ y el operando derecho igual a una expresión, que por conveniencia se le dará el nombre de:

$parRoto(X,Y) = roto(X)\ x\ \pi_Y(parece(X,Y))$

El operando izquierdo, que es una relación EDB, no requiere reglas. El operando derecho tiene como operador más externo a $x$, con el operando izquierdo que es una relación EDB y el operando derecho $\pi_Y(parece(X,Y))$. Esta última expresión puede ser transformada en una regla por el caso 3, que es:

$parecido(Y)$ :- $parece\ (X,Y)$ .

aquí se agregó el predicado nombrado como *parecido* para el predicado cuya relación es la misma que la expresión $\pi_Y(parece(X,Y))$.

Ahora, se puede escribir la regla para la expresión *parRoto*, usando el caso 4:

$parRoto(X,Y)$ :- $roto\ (X)$ & $parecido\ (Y)$.

finalmente, se usa el caso 2 para producir la regla para puedeComprar:

$puedeComprar(X,Y)$ :- $parece(X,Y)$ & $\neg parRoto(X,Y)$.

Hay que notar que las tres reglas desarrolladas son las mismas reglas producidas en el ejemplo 5.11.

## 6.1 De la lógica al álgebra.

Para cada programa no recursivo de datalog, cada relación IDB puede ser computada por una expresión equivalente del álgebra relacional. Esencialmente, todas las ideas para la construcción de la expresión algebraica

PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

deseada de una colección de reglas no recursivas ya han sido dados. Ahora solamente se ponen juntas apropiadamente:

> **Teorema 6.2**: sea $Z$ una colección de reglas no recursivas de datalog y seguras, posiblemente, con submetas negadas. Entonces, para cada predicado $p$ de $Z$ existe una expresión de Álgebra relacional que computa la relación para $p$.

Ya que $Z$ no es recursiva, se pueden ordenar los predicados de acuerdo a un orden topológico del grafo de dependencias; esto es, si $q$ aparece como una submeta en una regla para $p$, entonces, $q$ precede a $p$ en el orden. Esencialmente, se aplica el algoritmo 5.2 para evaluar la relación para cada predicado en su turno. Sin embargo, como ahora se tiene la posibilidad de submetas negadas, primero se usan los trucos del algoritmo 5.6 para reemplazar las relaciones de $R$ para submetas negadas por las relaciones complementarias $R' = DOM^k - R$, donde $k$ es el grafo de $R$ y $DOM$ es el conjunto de todos los símbolos que aparecen en $Z$ y en la relaciones EDB.

El conjunto $DOM$ siempre puede ser expresado en Álgebra relacional; esta es la unión de un conjunto de constantes y proyecciones de las relaciones EDB. También, la construcción del algoritmo 5.2 usa solamente los operadores de álgebra relacional. Como se debe componer estas operaciones algebraicas en expresiones con tantos operadores como sean necesarios, fácilmente se puede mostrar por inducción en el orden en el que los predicados son considerados que cada uno tiene una relación definida por alguna expresión del Álgebra relacional.

**Ejemplo 6.3**: considérense las siguientes reglas

$p(X) :- r(X,Y) \ \& \ \neg s(Y).$
$q(Z) :- s(Z) \ \& \ \neg p(Z).$

Asumir que $r$ y $s$ son predicados EDB con relaciones $R$ y $S$; se deben derivar expresiones para las relaciones $P$ y $Q$, que corresponde a los predicados IDB $p$ y $q$, respectivamente. La expresión algebraica para $DOM$ es la proyección de $R$ sobre el primero y segundo componente, más la relación unaria $S$, que es:

$$DOM = \pi_1(R) \ \cup \ \pi_2(R) \ \cup \ S$$

Se debe usar el orden topológico $p$, $q$. El predicado $p$ está definido por la primera regla. Para la primera submeta se puede usar la relación EDB $R(X,Y)$ y para la segunda submeta se usa la relación complementaria $[DOM - S](Y)$, por ejemplo, la relación unaria $DOM - S$ considerada como una relación sobre el atributo $Y$. Como es requerido por el algoritmo 5.2, se toma la unión de estas relaciones y la proyección sobre el atributo $X$, el único atributo de la cabeza. La expresión resultante es

$$P(X) = \pi_X(R(X,Y) \ |X| \ [DOM - S](Y)) \qquad \text{(i)}$$

Ahora, se construye la expresión para $Q$ de acuerdo a la regla (2). Para la primera submeta de la regla (2) se usa la relación $S(Z)$. Para la segunda submeta se necesita $[DOM - P](Z)$. Así, la relación $Q$ es $S(Z) \ |X| \ [DOM - P](Z)$, o, ya que el *join* es una intersección, en este caso,

$Q(Z) = [S \cap (DOM - P)](Z)$

Ya que $s$ es un subconjunto de $DOM$, lo anterior se simplifica a $Q(Z) = S(Z) - P(Z)$, o, sustituyendo (i), con $Z$ en lugar de $X$ para $P(Z)$,

$$Q(Z) = S(Z) - \pi_2(R(X,Y) \; |X| \; [DOM - S](Y))$$

Además, se puede decir que $DOM$ puede ser reemplazado por $\pi_2(R)$ en la expresión anterior. La razón es que $[DOM - S](Y)$ está unida con $R(Z,Y)$, así que sólo esos elementos de $DOM$ que están derivados del segundo componente de $R$ pueden contribuir al *join*.

## 6.2 Álgebra relacional monótona.

Hay que recordar que las cinco operaciones básicas del Álgebra relacional son monótonas con excepción de la diferencia. Las operaciones de unión, producto, selección y proyección pertenecen al subconjunto monótono del álgebra relacional. También se debe incluir en este subconjunto monótono cualquier operación derivable de estos cuatro operadores. Finalmente, la intersección aunque fue definida usando la diferencia, es, en efecto, un operador monótono.

El examen de las construcciones en los teoremas 6.1 y 6.2 indican que cuando no existen operadores de diferencia en la expresión algebraica, las reglas construidas por el teorema 6.1, no usan submetas negadas y cuando no hay submetas negadas, el teorema 6.2 provee una expresión usando solamente el subconjunto monótono del álgebra relacional. Entonces, se tiene la siguiente equivalencia.

> **Teorema 6.3:** el conjunto de funciones de las relaciones a relaciones expresables en el subconjunto monótono del álgebra relacional es el mismo que las funciones expresables en programas de datalog no recursivos que no incluyan submetas negadas.

## 6.3 Comparación entre datalog y el Álgebra relacional.

No es difícil mostrar que sin submetas negadas, pero con recursividad permitida, los programas de datalog son monótonos. Se puede mostrar que cualquier paso del algoritmo 5.3 que evalúa los programas de datalog, es una operación en el subconjunto monótono del álgebra relacional.

Si se acepta la monotonía de datalog sin la negación, entonces, fácilmente se puede ver que existen expresiones del Álgebra relacional, tales como $R - S$, que no son equivalentes a cualquier programa de datalog sin negación. Esto es, $R - S$ no es monótono, porque al agregar tuplas a $S$ decrementará el conjunto de tuplas en $R - S$, si alguna de las tupla agregadas está en $R$.

Similarmente, existen funciones que pueden ser expresadas en datalog recursivo, hasta sin negación, que no pueden ser expresadas en Álgebra relacional. Un ejemplo es la cerradura transitiva de una relación.

## 6.4 Calculo relacional.

Una forma de lógica llamada cálculo relacional conocido comercialmente como los lenguajes de búsquedas están basados en el modelo relacional. Esencialmente, el cálculo relacional es lo que se obtiene cuando se toma un programa de datalog no recursivo y se sustituye el OR lógico de los cuerpos de las reglas para todos los predicados, excepto al predicado que representa la respuesta de la búsqueda. Aquí se definirá la notación del cálculo relacional en una de sus dos formas llamada dominio del calculo relacional y tupla del cálculo relacional que será definida, posteriormente.

Cuando se introduzca la notación apropiada de seguridad, estas dos versiones de cálculo se pueden mostrar equivalentes en el poder expresivo de datalog no recursivo y, por lo tanto, al Álgebra relacional. La ubicuidad de esta clase de búsquedas está reforzada fuertemente por el hecho de que los lenguajes de bases de datos relaciónales comerciales tienen esencialmente la misma expresividad, aunque en ellos se encuentren ciertas capacidades agregadas (promedios, conteos, etc.) operaciones aritméticas. Por lo que ha sido común llamar completo a un lenguaje que es capaz de expresar, por lo menos, todas las búsquedas del Álgebra relacional.

### 6.4.1 Fórmulas del cálculo relacional.

Las fórmulas son expresiones que denotan relaciones, posiblemente infinitas. Cada fórmula tiene un conjunto de variables libres, que son análogas a las variables declaradas globalmente para un procedimiento. Otras variables que aparecen en una fórmula pueden estar limitadas y corresponden a variables locales de un procedimiento. El esquema de relación para una fórmula es un conjunto de atributos correspondientes a las variables libres de la fórmula.

Así, cómo con las variables locales y globales de los procedimientos, es posible para dos ocurrencias de la misma variable llamada $X$ referirse a dos declaraciones diferentes de $X$ y es posible que una ocurrencia esté limitada mientras que otra esté libre. Esto es, en general se debe distinguir entre las ocurrencias limitadas de las ocurrencias de variables libres. Así como se define recursivamente la capacidad de las fórmulas del cálculo relacional, al mismo tiempo se debe distinguir aquellas ocurrencias de variables que están libres de las que están limitadas.

1. Cada literal $p(X_1,\ldots,X_n)$, donde $p$ es un símbolo de predicado y $X_1,\ldots,X_n$ son variables o constantes , es una fórmula. El predicado $p$ representa una relación y la relación definida por esta literal es exactamente la misma que la relación definida por el algoritmo 5.1 para una regla cuyo cuerpo consiste de la submeta simple $p(X_1,\ldots,X_n)$. Todas las ocurrencias de variables entre $X_1,\ldots,X_n$ son libres.

2. Cada comparación aritmética $X\Theta Y$ es una fórmula, donde $X$ y $Y$ son variables o constantes y $\Theta$ es uno de los seis operadores aritméticos de comparación. Se toman en cuenta las ocurrencias de $X$ y $Y$, si son variables, como libres en la fórmula $X\Theta Y$. En algunos casos, $X\Theta Y$ representa una relación infinita, el conjunto de todos los pares $(X,Y)$ que están en la relación $\Theta$. Así, como en las reglas de datalog, se requiere de que tales fórmulas estén unidas por un operador *AND* lógico, a otra fórmula que defina una relación finita; en tal caso, la fórmula $X\Theta Y$ puede ser vista como una operación de selección.

Los incisos (1) y (2) arriba, forman las bases de la definición de fórmulas; las fórmulas que ellas describen son algunas veces llamadas *fórmulas atómicas.* En los siguientes incisos se dan las partes inductivas de la definición.

3. If $F_1$ y $F_2$ son fórmulas, entonces, $F_1 \Lambda F_2$ es una fórmula, con el significado intuitivo " ambas $F_1$ y $F_2$ son verdaderas". También, $F_1$ V $F_2$ es una fórmula que significa "por lo menos una de $F_1$ y $F_2$ son verdaderas" y $\neg F_1$ es una fórmula, que significa "$F_1$ no es verdadera". Las ocurrencias de variables son libres o limitadas en $F_1 \Lambda F_2$, $F_1$ V $F_2$ y $\neg F_1$ si y sólo si son libres o limitadas en cualquiera de $F_1$ y $F_2$ donde ocurra. Hay que notar que una variable $X$ podría estar limitada en algunas ocurrencias y libre en otras. En particular, se debe encontrar que $X$ está libre en $F_1$ y limitada en $F_2$, y que $X$ esté en $F_2$ no tiene influencia sobre el estado de ocurrencias de $X$ en $F_1$ ( o vice versa).

4. Si $F$ es una fórmula en donde $X$ aparece libre en por lo menos una ocurrencia, entonces $(\neg\exists X)F$ es una fórmula con el significado intuitivo de que existe por lo menos un valor de $X$ que cuando se sustituye por todas las ocurrencias libres de $X$ en $F$ hacer que la fórmula resulte verdadera. Se lee $(\neg\exists X)$ como existe una $X$. $(\neg\exists X)$ y $(\forall X)$, definidos anteriormente son llamados cuantificadores. Los cuantificadores juegan el rol de declaraciones tanto como les concierne la distinción de libre y limitada. Todas las ocurrencias libres de $X$ en $F$ están limitadas por el cuentificador $(\neg\exists X)$ y se consideran limitadas en la fórmula $(\neg\exists X)F$.

5. Si $F$ es una fórmula con, por lo menos, una ocurrencia libre de $X$, entonces, $(\forall X)F$ es una fórmula con el significado intuitivo de que cualquier valor que obtenga, si se sustituye ese valor por todas las ocurrencias libres de $X$ en $F$, entonces, la fórmula resultante viene a ser verdadera. Se lee $(\forall X)$ como "para toda $X$". Así como $(\neg\exists X)$, el cuantificador $(\forall X)$ contiene todas las ocurrencias libres de $X$ en $F$, así que estas ocurrencias están limitadas en $(\forall X)F$.

6. Si $F$ es una fórmula, lo es $(F)$ y su significado es el mismo que de $F$. Las ocurrencias de las variables están libres o limitadas en $(F)$ exactamente como están limitadas en $F$. Se necesitan paréntesis para la agrupación de operadores en fórmulas exactamente como se hace en otro tipo de expresiones. El orden de evaluación que se debe usar en la ausencia de paréntesis es:

   i) los operadores unarios, como prefijos, $\neg, (\forall X), (\neg\exists X)$ son de mayor precedencia y están agrupados a la derecha cuando aparece consecutivamente;

   ii) $\Lambda$ es el siguiente operador con mayor precedencia y se agrupa por la izquierda;

   iii) V tiene la menor precedencia y se agrupa por la izquierda.

Así, por ejemplo, la fórmula $(\forall X)\neg p(X,Y) \vee q(Y) \Lambda r(X)$ se agrupa como si estuviera parentizada

$$((\forall X)(\neg p(X,Y))) \vee (q(Y) \Lambda r(X))$$

Nótese que la ocurrencia de *X* en *r(X)* es libre, más bien que limitada por el cuantificador (∀X).

**Ejemplo 6.4:** si se tiene la fórmula de Álgebra relacional:

$\pi_{CUST}(\sigma_{INAME = 'Brie'}(INCLUDES\ |X|\ ORDERS))$

involucrando las relaciones INCLUDES (O#,INAME,QUANTITY) y ORDERS(O#,DATE,CUST)

En el cálculo relacional, el operador *join* se refleja por el operador lógico AND, con variables elegidas para corresponder con los atributos de las relaciones que están siendo operadas; esta es la misma idea que es usada para expresar el *join* en reglas de datalog. Así se puede iniciar con las fórmulas *includes(N,I,Q)* y *orders(N,D,C)*, correspondientes a las relaciones INCLUDES y ORDERS, respectivamente, con la variable *N* usada en ambas ocurrencias del atributo O#. Entonces:

INCLUDES |X| ORDERS

es representado por la fórmula

*includes (N,I,Q)* ∧ *orders(N,D,C)*

La selección para ITEM = 'Brie' es manejada por el operador lógico AND, con una tercera fórmula atómica, *I= 'Brie'*. Entonces,la proyección sobre el atributo CUST indica el interés solamente en la existencia de algún valor para cada una de las variables *N*, *I*, *Q* y *D* que hace a la fórmula

*includes (N,I,Q)* ∧ *orders(N,D,C)* ∧ *I='Brie'*

verdadera. Así se pueden cuantificar, existencialmente, estas cuatro variables, dejando a C o CUSTOMER como la única variable libre:

$$(\exists N)\,(\exists I)\,(\exists Q)\,(\exists D)\,(includes(N,I,Q) \wedge orders\ (N,D,C) \wedge I = 'Brie'\ ) \qquad (i.1)$$

Ya que uno de los conjuntos en (i.1) requiere que *I* sea igual a 'Brie', la única *I* que puede existir para hacer la fórmula verdadera es 'Brie'. Así se puede remover el cuantificador (∃I) si se reemplazan las ocurrencias de *I* por la constante 'Brie'. Con un pequeño esfuerzo se puede probar que (i.1) es equivalente a:

$$(\exists N)\,(\exists Q)\,(\exists D)\,(includes(N,'Brie',Q) \wedge orders\ (N,D,C))$$

en el sentido de que ambas fórmulas producen la misma relación sobre *C*, por ejemplo, el mismo conjunto de *customers*, cuando se dan las mismas relaciones de INCLUES y ORDERS.

### 6.4.2 Dominio del cálculo relacional.

Las fórmulas pueden ser usadas para expresar búsquedas en una forma simple. Cada fórmula con una o más variables libres definen una relación cuyos atributos corresponden a esas variables libres. Convencionalmente, se escribirá $F(X_1,\ldots,X_n)$ para significar de que la fórmula *F* tiene las variables

libres $X_1,\ldots,X_n$ y no otras. Entonces, la búsqueda o expresión denotada por $F$ es

$$\{X_1,\ldots X_n \mid F(X_1,\ldots,X_n)\} \qquad (i.2)$$

esto es, el conjunto de tuplas $a_1,\ldots,a_2$ tal que cuando se sustituye $a_i$ por $X_i$, $1<i\leq n$, la fórmula $F(a_1,\ldots,a_n)$ se hace verdadera.

El lenguaje de búsquedas consistente de expresiones en la forma de (i.2) se llama Dominio del Cálculo Relacional (DCR). El adjetivo *dominio* parece extraño en este contexto, pero, se refiere al hecho de que las variables son componentes de tuplas, por ejemplo, las variables están como miembros arbitrarios del dominio de sus componentes. Esta forma de lógica debe ser distinguida de tupla del cálculo relacional, donde las variables están para todas las tuplas .

Se debe observar que las relaciones definidas por expresiones DCR, necesitan no ser finitas. Por ejemplo

$$\{XY \mid \neg p(X,Y)\}$$

es una expresión DCR legal definiendo el conjunto de pares $(X,Y)$ que no están en la relación de predicado $p$. Para evitar tales expresiones se introducirá más adelante el subconjunto de fórmulas DCR llamadas seguras, en analogía con reglas seguras de datalog.

### 6.4.3 Del Álgebra relacional al dominio del Cálculo relacional.

Antes de tratar la cuestión de seguridad, se mostrará cómo expresar Álgebra relacional en DCR. Las ideas están cerca de las del proceso de trasladar el Álgebra en reglas de datalog. Únicamente el operador de unión necesita ser manejado en diferente forma, porque mientras datalog permite el uso de varias reglas para expresar un "or" de posibilidad, en el cálculo relacional se necesita el operador lógico OR.

> **Teorema 6.4:** cada búsqueda expresable en Álgebra relacional es expresable en un dominio del cálculo relacional.

Prueba: la prueba es una inducción sobre el número de operadores en la expresión algebraica. Se muestra que para cada expresión $E$ del álgebra relacional definiendo una relación de grado $k$, existe una fórmula $F(X_1,\ldots,X_k)$ de DCR definiendo la misma relación. Las bases, cero operadores, cubre el caso donde $E$ es una simple relación $R$, o una relación constante. Si hay una relación llamada $R$, entonces, $R$ debe ser de grado $k$ y se debe usar el predicado $r$ para representar $R$ en la forma de DCR. Entonces, la fórmula deseada es solamente $r(X_1,\ldots,X_k)$.

La otra parte de las bases es más compleja de situar, pero, la idea es simple. Supóngase por conveniencia que la relación constante tiene solamente dos tuplas: $a_1,\ldots,a_k$ y $b_1,\ldots,b_k$. La generalización a cualquier número finito de tuplas debería ser obvio cuando se observa que los miembros en el conjunto finito anterior está expresado por la fórmula:

$$(X_1= a_1 \wedge \ldots \wedge X_k=a_k) \vee (X_1=b_1 \wedge \ldots \wedge X_k=b_k)$$

Para la inducción, se consideran los cinco casos correspondientes a las cinco operadores básicos del álgebra relacional.

Caso 1: $E = E_1 \ U \ E_2$. Se debe asumir que $E$, $E_1$ y $E_2$ son todas de grado $k$. Por la hipótesis inductiva existen fórmulas DCR $F_1$ y $F_2$ que definen las relaciones de $E_1$ y $E_2$ respectivamente. por la sustitución de las variables libres en una de las fórmulas, se puede asumir que $F_1$ y $F_2$ ambas tienen $X_1,\ldots,X_k$ como sus variables libres y que estas variables corresponden a los componentes de las tuplas de $E$, $E_1$ y $E_2$ en ese orden particular, $X_1,\ldots X_k$. Entonces, la fórmula para $E$ es $F_1 \vee F_2$.

Caso 2: $E = E_1 - E_2$ . Como en el caso 1, se asume que existen las fórmulas $F_1(X_1,\ldots,X_k)$ y $F_2(X_1,\ldots,X_k)$ que produce las relaciones de $E_1$ y $E_2$ respectivamente. Entonces, la fórmula para $E$ es $F_1 \wedge \neg F_2$.

Caso 3: $E = \pi_{i1},\ldots,\pi_{ik}(E_1)$. Sea la relación de $E_1$ de grado $n$. Por la hipótesis inductiva existe una fórmula $F_1(X_1,\ldots,X_n)$ que produce la misma relación que $E_1$. sea $j_1,\ldots,j_{n-k}$ la lista de $(1,\ldots,n)$ que no aparece entre $i_1,\ldots,i_k$. Entonces, una tupla $\mu$ está en la relación de $E$ si y sólo si existen valores de los componentes $j_1,\ldots,j_{n-k}$ con el cual se puede extender $\mu$ y de ese modo se produce una tupla de $E_1$. En términos del cálculo relacional se dice:

$$F(X_{i1},\ldots,X_{ik}) = (\exists X_{j1})(\exists X_{j2})\ldots(\exists X_{jn-k})F1(X_1,\ldots,X_n)$$

para obtener la fórmula $F$ para $E$.

Caso 4: $E = E_1 \ X \ E_2$. Sean $F_1(X_1,\ldots,X_n)$ y $F_2(Y_1,\ldots,Y_m)$ fórmulas para $E_1$ y $E_2$ respectivamente. Cambiar de nombre a las variables si es necesario, se debe asumir que $X$ y $Y$ no tienen variables en común. Entonces:

$$F(X_1,\ldots,X_n,Y_1,\ldots,Y_m) = F_1(X_1,\ldots,X_n) \wedge F_2(Y_1,\ldots,Y_m)$$

es una fórmula para $E$ .

Caso 5: $E = \sigma_A(E_1)$. Se asume que $A$ es una selección simple de la forma $\$_i\Theta\$_j$ o $\$_i\Theta a$. Por la hipótesis inductiva, existe una fórmula $F_1(X_1,\ldots,X_k)$ para $E_1$. Entonces la fórmula $F_1 \wedge X_i\Theta X_j$ o $F_1 \wedge X_i\Theta a$ sirve para $E_1$ dependiendo de la forma de $A$.

**Ejemplo 6.5:** siguiendo los pasos del ejemplo 6.2 para producir una expresión DCR para la expresión algebraica:

$$parece(X,Y) - (roto(X) \ x \ \pi_Y(parece(X,Y)))$$

para el operando izquierdo de la diferencia, se tiene la fórmula DCR $parece(X,Y)$.

Se debe trabajar sobre el operando derecho de -, cuyo operador más externo es $x$. Ese operador, en turno, tiene un operando izquierdo con las formula DCR $roto(X)$. Su operando derecho se obtiene aplicando el caso 3 a la fórmula $parece(X,Y)$, produciendo $(\exists X)parece(X,Y)$.

La última fórmula tiene la variable libre $Y$, y, la fórmula $roto(X)$ para el operando izquierdo de $x$ tiene la variable libre $X$ así que pueden ser combinadas para obtener la fórmula para el lado derecho del operando de -, que es:

$roto(X) \wedge (\exists X) parece(X,Y)$

Hay que notar que la ocurrencia de $X$ en roto es libre mientras la ocurrencia de $X$ en *parece* está limitada por un cuantificador existencial.

La fórmula para la expresión algebraica completa se obtiene por el caso 2 de $parece(X,Y)$ y la fórmula anterior. Se puede convertir en una búsqueda al insertarla en un conjunto formado, como:

$$\{XY \mid parece(X,Y) \wedge \neg(roto(X) \wedge (\exists Z) parece(Z,Y))\}$$

Para mayor claridad, se ha reemplazado la ocurrencia limitada de $X$ por $Z$. Así como las variables locales en los programas, se pueden reemplazar todas la ocurrencias de una simple variable limitada como $X$ por otra variable, como $Z$, que no aparece libre en el alcance del cuantificador que declara $X$.

### 6.4.4 Fórmulas de dominio independiente.

Para llegar al concepto de fórmulas seguras, se debe considerar, primero, lo que es necesario considerar concerniente a las expresiones del cálculo relacional. Esta propiedad llamada *independencia de dominio* es una noción semántica.

Para cada variable libre de una fórmula, se puede imaginar que existe algún dominio de posibles valores que esta variable puede asumir. Existen ciertos valores que tendrán sentido para excluirlos del dominio. Los valores que aparecen en la misma fórmula y los valores que aparecen en las relaciones dadas. Se llama a este conjunto $DOM(F)$, donde, $F$ es una fórmula. Hay que notar que $DOM(F)$ no es una constante dependiendo sólo sobre $F$, pero más bien una fórmula de las relaciones que son dadas por los predicados de $F$. $DOM(F)$ es una simple función; siempre puede ser expresada en álgebra relacional como la unión del conjunto de constantes que aparecen en $F$ y las proyecciones de todas la relaciones que son argumentos de $F$ proyectadas sobre cada uno de sus componentes. Por supuesto, la relación $DOM(F)$ es de grado 1.

**Ejemplo 6.6:** considérese la fórmula

$$F = p(X,Y) \wedge q(Y,Z) \vee X > 10$$

Tengan los predicados $p$ y $q$ los relaciones $P$ y $Q$ respectivamente. Entonces,

$$DOM(F) = \{10\} \cup \grave{O}_1(P) \cup \grave{O}_2(P) \cup \grave{O}_1(Q) \cup \grave{O}_2(Q)$$

Intuitivamente, si la relación definida por una fórmula $F$ incluye tuplas con símbolos que no están en $DOM(F)$, entonces, de alguna forma, la fórmula no está atendiendo a los datos que se le han dado, pero, está tomando datos de algún otro lugar.

### 6.4.5 Fórmulas RCD seguras.

No existe algoritmo alguno que indique cuando una fórmula RCD tiene un dominio independiente. Así, los lenguajes de búsqueda reales basados en el cálculo relacional usan sólo un subconjunto de las fórmulas RCD, las que tienen un dominio independiente garantizado. Se definen fórmulas seguras al subconjunto de fórmulas de dominio independiente. Las propiedades de seguridad importantes son:

a) cada fórmula segura debe tener un dominio independiente;

b) se puede decir fácilmente, por inspección, si una fórmula es segura o no;

c) las fórmulas que se pueden expresar en lenguajes de búsquedas basados en el cálculo relacional, son seguras.

Con este criterio en mente, se define una regla segura como:

1. no existe el uso del cuantificador $\forall$. Esta condición no afecta a la expresividad del lenguaje, porque $(\forall X)F$ es, lógicamente, equivalente a $\neg(\exists X)\neg F$. Esto es, $F$ es verdadera para todas las $X$ si y sólo si no existe una $X$ para la que $F$ sea falsa. Aplicando esta transformación se pueden eliminar los cuantificadores universales;

2. cuando un operador OR es usado, las dos fórmulas conectadas, $F_1$ V $F_2$, tiene el mismo conjunto de variables libres; es decir, que son de la forma:

$$F_1(X_1,\ldots,X_n) \ \text{V} \ F_2(X_1,\ldots,X_2)$$

3. considérese cualquier subfórmula consistente de la conjunción de una o más fórmulas $F_1 \Lambda \ldots \Lambda F_m$. Entonces, todas la variables que aparecen libres en cualquiera de las fórmulas debe está limitada por :

   a) una variable está limitada si es libre en alguna $F_i$, donde $F_i$ no es una comparación aritmética y no está negada;

   b) si $F_i$ es $X = a$ o $a = X$, donde $a$ es una constante, entonces, $X$ está limitada;

   c) si $F_i$ es $X = Y$ o $Y = X$, y $Y$ es una variable limitada, entonces, $X$ está limitada.

   Es importante notar que esta regla se aplica a fórmulas atómicas si no aparecen en un grupo largo de fórmulas conectadas por AND. Por ejemplo, si la fórmula completa es $X = Y$ , no es segura, porque es el conjunto de una fórmula y las variables $X$ y $Y$ no están limitadas. Lo mismo pasa con la fórmula $X = Y$ V $p(X,Y)$ ya que el operador izquierdo del OR viola la regla (2). Sin embargo $X = Y \Lambda p(X,Y)$ es segura, porque $p(X,Y)$ limita a ambas por (a);

4. un operador $\neg$ se puede aplicar sólo a un término en una conjunción del tipo discutido en (3). En particular, una subfórmula $\neg G$ viola la definición de seguridad a menos que sea parte de una subfórmula más larga

$$H_1 \Lambda \ldots \Lambda H_i \Lambda \neg G \Lambda I_1 \Lambda \ldots \Lambda I_j.$$

### 6.4.6 Cálculo relacional de tupla.

El cálculo relacional de tupla o CRT, es una variante del cálculo relacional donde las variables representan las tuplas en lugar de componentes

de tuplas. Para hacer referencias al *i*_ésimo componente de una tupla $\mu$ se usa $\mu[i]$. Si un atributo *A* para ese componente se conoce, se puede escribir $\mu[A]$.

Las fórmulas CRT están definidas recursivamente y la estructura de las fórmulas CRT es bastante cercana a la estructura de las fórmulas DCR. las bases de las fórmulas atómicas son:

1. si *p* es un predicado nombrado y $\mu$ es una variable de tupla, entonces $p(\mu)$ es una fórmula atómica. Su relación es que la tupa $\mu$ esta en la relación para *p;*

2. *X Ú Y* es una fórmula atómica si *Ú* es un operador de comparación aritmético y *X* y *Y* son ambas constantes o referencias de componentes.

La definición intuitiva procede como un dominio de cálculo relacional. Si $F_1$ y $F_2$ son fórmulas de CRT y $\mu$ es una variable de tupla que aparece libre en $F_1$, entonces, las siguientes son fórmulas CRT, con su obvio significado y conjunto de variables libres y limitadas:

a) $F_1 \wedge F_2$
b) $F_1 \vee F2$
c) $\neg F_1$
d) $(\exists \mu) F_1$
e) $(\forall \mu) F_2$

La relación asociada con una fórmula de CRT se define de la misma manera que una DCR. LA relación para *F* tiene un componente para cada componente de cada variable de tupla libre de *F*. El valor de la relación para *F* es el conjunto de las tuplas cuyos valores, cuando se sustituyen por sus componentes correspondientes a variables de tupla, *F* se hace verdadera. Una búsqueda de CRT es una expresión e la forma $\{\mu | F(\mu)\}$, donde $\mu$ es la única variable libre de *F*. Naturalmente, esta expresión define la relación de todas las tuplas $\mu$ tal que $F(\mu)$ sea verdadera.

En ocasiones, el grado de la variable de la tupla no será clara del contexto. Se usará la notación $\mu^{(i)}$ para denotar que la variable de tupla $\mu$ es de grado *i* .

**Ejemplo 6.7:** la búsqueda DCR derivada en el ejemplo 6.5 se puede convertir a CRT si se usa la tupla $\mu$ para *(X,Y)*, *v* para *X* en *roto* y *p* para *(Z,Y)* en *parece*. La búsqueda es:

$$\{\mu^{(2)} \mid parece(\mu) \wedge \neg((\exists v^{(1)})(roto(v) \wedge v[1] = \mu[1] \wedge (\exists p^{(2)})(parece(p) \wedge p[2] = \mu[2]))\}$$

Nótese cómo la fórmula atómica $v[1] = \mu[1]$ reemplaza la conexión que en DCR fue expresada usando la misma variable *X* en *parece* y en *roto*. no se puede usar $\mu$ como el argumento de *roto*, porque *roto* toma una tupla unaria, mientras que *parece* toma una tupla binaria. Similarmente, $p[2] = \mu[2]$ sustituye para el uso doble de *Y*. Nótese también que *v* debe estar existencialmente cuantificada o si no debe ser una variable libre de la búsqueda. Aunque pareciera que *v* puede ser cualquier cosa, la fórmula

$v[1]=\mu[1]$ define completamente la tupla unaria $v$. La relación para la fórmula $roto(v) \wedge v[1] = \mu[1]$ es"

$\{(X,Y) \mid X$ está en $roto\}$

Los dos componentes correspondientes para $v[1]$ y $\mu[1]$. tienen el mismo valor en cada tupla, porque en orden la fórmula que sea satisfecha, la subfórmula $v[1]=\mu[1]$ debe ser satisfecha.

En la fórmula $(\exists v^{(1)})\,(roto(v) \wedge v[1] = \mu[1])$ existe sólo un componente, $\mu[1]$. la relación para esta fórmula es :

$\{(X,Y) \mid X$ está en $roto\}$

es la misma que para *roto*.

### 6.4.7 Tuplas seguras del cálculo relacional.

Como en DCR, se pueden escribir fórmulas CRT , tales como $\neg r(\mu)$, que denotan relaciones infinitas. Se quiere evitar este tipo de fórmulas en lenguajes de búsquedas reales y así como con CRD, la aproximación más usual es definir una forma restringida de CRT llamada CRT segura. Nuevamente en analogía con RCD se debe ser más restrictivo que cuando se define seguridad, porque todo lo que se necesita hacer es definir una clase que refleje lo que se encuentra en los lenguajes comerciales basados en CRT y que es equivalente en el poder expresivo del álgebra relacional.

Ya que el grado de la variable de tupla no siempre está claro del contexto en una fórmula CRT, se asumirá que el grado de cada variable está dado y que el grado de una variable no cambia de ocurrencia a ocurrencia, aunque si dos ocurrencias están limitadas por cuantificadores diferentes. La seguridad de una fórmula CRT se define como sigue:

1. so se usa el cuantificador $\forall$;

2. cuando se usa el operador $\vee$, las dos fórmulas conectadas, $F_1 \vee F_2$, tiene sólo variables de tuplas libres y es la misma variable;

3. considérese cualquier subfórmula consistente de una conjunción de una o más fórmulas $F_1 \wedge \ldots \wedge F_m$. Entonces, todos los componentes de las variables de tuplas que son libres, en alguna de las $F_i$ están limitadas en el siguiente sentido:

   a) si $F_i$ no está negada, no es una comparación aritmética y tiene una variable de tupla libre $\mu$, entonces, todos los componentes de $\mu$ están limitados;

   b) si $F_i$ es $\mu[j] = a$ o $a = \mu[j]$, donde $a$ es una constante, entonces, $\mu[j]$ es limitada;

   c) si $F_i$ es $\mu[j] = v[k]$ o $v[k] = \mu[j]$, y $v[k]$ es una variable limitada, entonces, $\mu[j]$ está limitada;

4. un operador $\neg$ debe ser aplicado sólo a un término en una conjunción del tipo discutido en la regla(3).

## 6.5 Suposición de mundo cerrado.

Se ha visto cómo la lógica puede ser usada para definir operaciones sobre una base de datos y cómo el proceso de obtener respuestas de una búsqueda es una de las pruebas de ciertos datos seguidos, lógicamente, de los datos en la base de datos extensional. Sin embargo, seguir consistentemente con el punto de vista de que las operaciones sobre los datos son pruebas, se debe ser capaz de probar que algunos datos no son parte de la respuesta de la búsqueda probando que sus negaciones siguen de la EDB y las reglas dadas.

Desafortunadamente, no existe nada en la lógica que permita concluir un hecho negativo. Por ejemplo, supóngase que se tiene una relación EDB llamada *P* para el predicado *padre*, se debe suponer que no existen en *P* tuplas que permitan usar la regla (1) de la figura 4.1, que es:

$$hermano(X,Y) := padre(X,Z) \ \& \ Padre(Y,Z) \ \& \ X \neq Y$$

para deducir $hermano(Carlos,Herman)$. Se puede, por lo tanto, deducir la verdad de la fórmula lógica:

$$\neg(hermano(Carlos,Herman)) \qquad (I)$$

Si se asume que no existe $Z$ para la que $pariente(Carlos,Z)$ y $pariente(Herman,Z)$ son ambas verdaderas, entonces, no hay forma de que

$$hermano(Carlos,Herman)$$

pueda ser deducida por la regla (1) de la figura 4.1. Pero ésta no es una prueba de que *(I)* sea verdadera. Es aquí donde se debe hacer la suposición de mundo cerrado (SMC), que dice que cuando una fórmula atómica $p(a1,\ldots,a_k)$ sin variables, no se puede deducir de su EDB o sus reglas, entonces se debe asumir $\neg p(a_1,\ldots,a_k)$. Así la SMC es una regla poderosa para inferir nuevas fórmulas. La SMC actúa como una metaregla que habla de autodeducciones. La SMC permite deducir hechos de la forma $\neg p(a_1,\ldots,a_k)$ cuando la forma usual de deducción no produce $p(a_1,\ldots,a_k)$.

## 7. LÓGICA CON FUNCIONES DE SÍMBOLOS

Hasta el momento se ha tratado a la lógica al estilo de Datalog, donde los argumentos de los predicados son solamente variables o constantes. Esto es perfectamente razonable cuando se trabaja con un lenguaje de búsquedas del modelo relacional que se puede visualizar como una notación no recursiva de Datalog. Existen, sin embargo, algunas situaciones descritas más fácilmente con el uso de argumentos que son *términos* construidos con variables, constantes y funciones de símbolos. Un término puede ser definido recursivamente por:

1. una variable es un término;
2. una constante es un término;
3. si $f$ es una *función de símbolos*, y $T_1,\ldots,T_k$ son términos, entonces, $f(T_1,\ldots,T_k)$ es un término

En primer orden lógico, los argumentos de predicados pueden ser términos arbitrarios, en lugar de ser únicamente variables o constantes. Se continúa con el uso del estilo de las reglas de Prolog, esto es, cláusulas de Horn, posiblemente con submetas negativas en el cuerpo. Las tuplas de las relaciones EDB podrán contener componentes que son arbitrariamente términos, en lugar de ser simplemente constantes.

Se asume que los términos en las relaciones EDB no tienen variables. Los términos sin variables se llaman *términos fundamentales.* Un predicado cuyos argumentos son términos fundamentales es un *átomo fundamental*. Se debe ver a un átomo fundamental como una tupla de una relación, pero los componentes de esas tuplas no son necesariamente valores elementales ( por ejemplo, enteros o caracteres); pueden ser términos con una estructura compleja.

**Ejemplo 7.1:** un uso simple de una función de símbolos es incorporar en las relaciones alguna de las estructuras que son posibles en el modelo jerárquico. Así, se puede almacenar la relación EDB *empleado* con el esquema

empleado(nombre(F,L), dirección(calle(N,S),ciudad(C,T,Z)))

aquí *empleado* es un símbolo del predicado con una relación binaria. El primer argumento es un término consistente de una función de símbolos binaria *nombre* aplicada a los argumentos $F$ y $L$ que representan el nombre y apellido del empleado. Nótese que las variables $F$ y $L$ serán, en cada tupla de la relación, reemplazadas por dos constantes apropiadas.

El segundo componente en una función de símbolos binaria *dirección* aplicada a dos argumentos, que no son términos triviales. El primero es una función de símbolos *calle* aplicada a dos variables $N$ y $S$ que contienen el número y el nombre de la calle respectivamente, mientras que el segundo es una función de símbolos *ciudad* aplicada a las variables $C$, $T$ y $Z$ que contienen el nombre del departamento, el nombre del municipio y el código postal. Una posible tupla de la relación *empleado* es

empleado(nombre(Omar,Díaz),dirección(calle(12,central),
ciudad(Guatemala,Guatemala,01000))

La tupla anterior tiene un esquema, para la relación empleado, la principal importancia de las funciones de símbolos es que se pueden permitir varios esquemas diferentes y aplicar apropiadamente la función de símbolos. Por ejemplo, si un empleado vive en un apartamento, entonces en lugar del término *calle(N,S)* en el primer componente de dirección, se puede tener un término



PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

*apto(A,calle(N,S)*, donde *A* es el número del apartamento. Así, si Ana Caal vive en un edificio de apartamentos, se puede tener la siguiente tupla

```
empleado(nombre(Ana,Caal),dirección(apto(4,calle(4,Sur)),
                ciudad(Guatemala,Guatemala,01000)))
```

Supóngase que se quiere imprimir la dirección en un formato personalizado: el nombre del empleado en la línea 1, el número del apartamento, si existe, en la siguiente línea, la dirección en la siguiente, el departamento, municipio y código postal en la última. Así la tercera línea contendrá la calle si el primer componente de la dirección tiene un número de apartamento, esto es, si la función de símbolos es *apto*. Si la función de símbolos es *calle*, entonces la tercera línea contendrá la información en el segundo componente de *dirección*. Se pueden expresar estas observaciones como reglas con funciones de símbolos. Las siguientes reglas definen un predicado *impresión3* que se debe imaginar como el conjunto de las terceras líneas de las direcciones de todos los empleados.

```
impresión3(N,S) :- empleado(X,dirección(apto(Y,calle(N,S)),Z)).
impresión3(C,T,Z) :- empleado(U,dirección(calle(V,W),
                                ciudad(C,T,Z))).
```

Nótese como la estructura que provee la función de símbolos no solamente permite deducir el significado de diferentes campos en una tupla, también permite ignorar la estructura de partes de tuplas en las que no se tiene interés. Por ejemplo *Z* en la primera regla corresponde a lo que es el campo de *ciudad* para la porción de *dirección* de la tupla.

formalmente la relación para *impresión3* se asume que debe tener un grado particular y puede ser que no tenga sentido colocar en esa relación tuplas de distinto grado. Sin embargo, lo que se intenta en este ejemplo es que todos los componentes de la tupla de *impresión3* sean impresos en el orden de apariencia; no existe una relación IDB construida para la relación *impresion3*. Para estar formalmente correcto, se deben agrupar los argumentos de impresión3 en una función de símbolos, ya que las funciones de símbolos pueden ser usadas con un número diferente de argumentos, esto quiere decir que se puede escribir el encabezado de la regla para impresión3 como *impresión3(línea(N,S))* e *impresión(línea(C,T,Z))*.

En el ejemplo 7.1, la función de símbolos se usa en efecto para crear un registro de estructura nombrado y para definir estructuras alternativas para los campos de los registros, esta capacidad puede ser en algunos casos una conveniencia. Por ejemplo se puede definir una relación ordinaria con el formato

```
empleado(F,L,A,N,S,C,T,Z)
```

donde *A* de apartamento es nulo para aquellos que no viven en un apartamento.

Cuando las reglas son recursivas, el uso de una función de símbolos permite crear un conjunto infinito de los valores potenciales que debe aparecer en la relación, como se muestra en el próximo ejemplo. Nótese que las reglas de Datalog no tienen el poder suficiente de crear valores que no existan en una relación EDB actualmente o en las mismas reglas.

**Ejemplo 7.2:** la adición puede ser definida recursivamente como sigue. Supóngase que se tiene el predicado de símbolos *int(X)* que trata de ser verdadero si *X* es un entero y un predicado *sum(X,Y,Z)*, que será verdadero si y sólo si la suma de *X* y *Y* es *Z*, supóngase también que se tiene la función de símbolos *sig* que tiene la intención de que *int(X)* es el entero *X+1*. Finalmente, se tiene la constante *0*. Entonces, el término *sig(0)* es la forma de decir "1", *sig(sig(0))*

es "2" y así sucesivamente; los enteros diferentes a 0 no existen de otra forma. Entonces, las reglas necesarias se definen en la figura 7.1.

```
(1) int(0)
(2) int(sig(X)) :- int(X)
(3) sum(X,0,X) :- int(X)
(4) sum(X,sig(Y),sig(Z)) :- sum(X,Y,Z)
```

**Figura 7.1 reglas para la definición de adición.**

Las dos primeras reglas en la figura 7.1, definen 0, sig(0), sig(sig(0)) y así sucesivamente, como enteros. La tercera regla indica que la suma de cualquier entero y 0 es un entero y la última indica que X+(Y+1)=Z+1, si X+Y=Z, escrito de otra forma X+(Y+1) = (X+Y)+1, estas reglas son suficientes para generar todas y solamente las adiciones verdaderas.

Por ejemplo, *int(0)* se toma de la regla (1). Entonces, de la regla (3), sum(0,0,0) es verdadera. Aplicando la regla(4) a estos datos se tiene que *sum(0,sig(0),sig(0))*, esto es *sum(0,1,1)*. De la regla (2) se infiere *int(sig(0))*, que es *int(1)*, entonces, la regla (3) da *sum(1,0,1)* y (4) da *sum(1,1,2)*. Con esto se ha usado 1 para *sig(0)* y 2 para *sig(1)=sig(sig(0))*. Nótese que tanto *int* como *sum* tienen relaciones infinitas, aunque las reglas de este ejemplo son seguras.

Hay que observar que aunque la sintaxis para la función de símbolos y predicado de símbolos es la misma ( las cadenas de caracteres inician con una minúscula), no existe confusión entre los dos tipos de símbolos. Las funciones de símbolos toman los valores del dominio como argumentos y regresan valores del dominio. Los predicados de símbolos toman los valores del dominio como argumentos, pero regresan valores de verdadero o falso. Así, en el ejemplo 7.2, *sig* toma un entero como un argumento y regresa un entero, mientras que *sum* toma tres enteros como argumentos y regresa verdadero si el tercer valor es la suma del primero y el segundo y regresa falso si es otro valor. En el ejemplo 7.2 se puede ver una función de símbolos como *nombre* tomando dos elementos como dominio y regresando un objeto con una estructura definida.

En el próximo ejemplo se muestra cómo las funciones de símbolos pueden ser usadas para una infinidad potencial de nuevos objetos y, también, para dar pistas que permitan descifrar la estructura del objeto. Es un ejemplo trivial del razonamiento de un sistema acerca de un mundo de cinco estados. Permite describir los hechos verdaderos acerca de las transiciones de estado a estado y permite planificar cómo ir de un estado a otro. En una situación real pueden existir millones de estados y el poder de los sistemas de bases de datos basados en la lógica debe ser necesario para deducir los hechos eficientemente.

**Ejemplo 7.3:** considérese un mundo simple de bloques con tres bloques, *A*, *B*, y *C* que deben colocarse de forma apilada solamente si la pila se encuentra en un orden decreciente. El bloque *A* es mayor que el bloque *B* que es mayor que el bloque *C*. Los cinco posibles estados del sistema se denotan desde la letra *a* hasta la *e* en la figura 7.2.

Los movimientos legales del sistema se expresan por una relación EDB IR(S,T) que es verdadera si y solo si el estado S se puede convertir en el estado T con un movimiento. Aquí un movimiento cambia la posición de un bloque, removiéndolo del bloque o del piso donde se encuentra el resto y colocándolo ya

sea en el piso o en un bloque más largo que el mismo. Para los estados de la figura 7.2, la relación IR consiste en 10 tuplas, que escritas como átomos son:

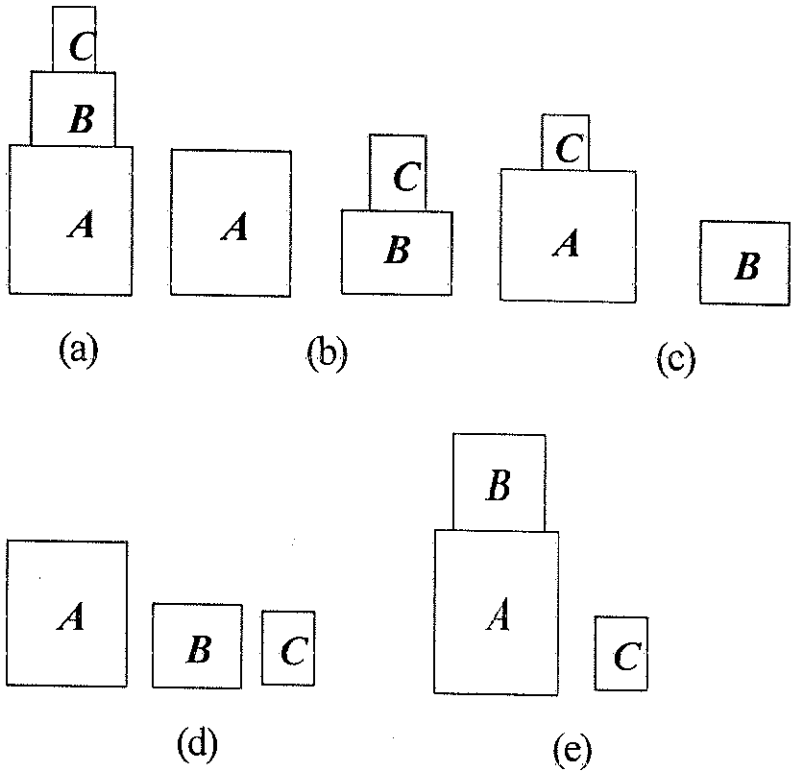



**Figura 7.2: Estado del mundo de bloques.**

```
ir(a,e) ir(b,c) ir(b,d) ir(c,b) ir(c,d)
ir(d,b) ir(d,c) ir(d,e) ir(e,a) ir(e,d)
```

Sea *g* una función de símbolos binaria y sea *g(S,T)* un término representando la posibilidad de que el estado *S* pueda ser transformado en el estado *T* por uno o más movimientos. Se define el predicado unario *verdadero* con la intención de que *verdadero(X)* encaje exactamente cuando el término *X* represente una sentencia acerca del estado de transición que es verdadero. Por ejemplo, se puede tener naturalmente la regla

```
verdadero(g(S,T)) :- ir(S,T).                    (1)
```

aquí, *ir* se presume que es un predicado EDB que corresponde a la relación EDB *IR*.

También se necesita una regla que defina los movimientos, ya sea uno o más, que es,

```
(2)   verdadero (g(S,T)) :- verdadero(g(S,U)) & verdadero(g(U,T)).
```

Ahora, se debe hablar de estados accesibles. Supóngase que se tiene la relación unaria EDB *ACC(S)* de estados *S* que son inicialmente accesibles. Por ejemplo *ACC* debe contener sólo el estado que inicialmente se encuentra en el mundo de bloques. Entonces, se debe usar la función de símbolos unaria *a(S)* que

representa la posibilidad de que el estado *S* sea accesible después de algún número de movimientos, las reglas para que una condición de accesible sea verdadera son:

```
(3)   verdadero(a(S)) :- acc(S).
(4)   verdadero(a(S)) :- verdadero(a(T)) & verdadero(g(T,S)).
```

En las reglas de la (1) a la (4) el uso de las funciones de símbolos no es esencial. Se puede remover "verdadero" en cada lugar donde aparece, tratando a *a* y *g* como si fueron predicados en lugar de funciones de símbolos y las reglas indicarán esencialmente la misma cosa. Sin embargo, existen otras cosas que se pueden decir acerca de las aproximaciones tomadas aquí, que son considerablemente más naturales cuando se usan funciones de símbolos. Por ejemplo, se debe usar términos que expresen condiciones más complicadas que "ir" y "accesible". Se puede, por ejemplo, usar una función de símbolos binaria *or* y *and* para representar dos condiciones lógicas, se pueden, entonces, incluir en la base de conocimientos las reglas definiendo *AND* y *OR*.

```
(5)   verdadero(or(X,Y)):- verdadero(X).
(6)   verdadero(or(X,Y)):- verdadero(Y).
(7)   verdadero(and(X,Y)):- verdadero & verdadero(Y)
```

Nótese que estas reglas no dependen de las estructuras de los términos X y Y, pueden ser la unión y disyunción de los valores de "ir" y "accesible". Nótese también que las reglas (5) y (6) no son seguras ya que cada una tiene una variable que aparece en la cabeza, pero, no en el cuerpo, existen dos posibles soluciones para resolver este problema.

1. Escribir otro posible predicado *posible(X)* que indique que *X* es un término que debe ser verdadero. Se puede entonces usar *posible(X)* para restringir el rango de la variable que de otra forma no aparece en el cuerpo de una regla. Por ejemplo la regla (5) puede ser escrita como:

   ```
   verdadero(or(X,Y)) :- verdadero(X) & posible(Y).
   ```

2. Se pueden aceptar las reglas no seguras. Ya que se puede prevenir el uso de una regla para inferir todos los datos verdaderos, ya que del hecho de que a *verdadero(g(a,e))* le siguen una infinidad de hechos con valores diferentes de Y en la regla (5). Sin embargo, no previene para responder respuestas acerca de valores particulares como " es *or(g(a,b,g(c,d))* verdadera?".

## 7.1 Evaluación de la lógica con funciones de símbolos.

En el algoritmo 5.3, llamado "evaluación de ecuaciones de Datalog" para procesar el punto fijo mínimo de una colección de reglas con submetas no negadas. La idea es iniciar asumiendo que las relaciones IDB se encuentran vacías y, entonces, repetidamente computar los nuevos valores para los predicados IDB aplicando las reglas a las relaciones EDB y los valores actuales de las relaciones IDB. Cuando las reglas son Datalog, este proceso eventualmente converge a una relación finita que es el significado de las reglas. Cuando existen funciones de símbolos se puede mostrar que un punto fijo mínimo existe pero este puede ser infinito, en tal caso, no existe un número finito de repeticiones, del ciclo de la evaluación, suficiente para alcanzar el punto fijo mínimo.

En el capítulo 5 se explicó el algoritmo 5.4 llamado evaluación semi ascendente de ecuaciones de Datalog, en dicho algoritmo cuando se encuentra un

valor, se encuentra nuevamente en los pasos subsecuentes, la evaluación semi ascendente evita descubrir el mismo valor de la misma manera dos veces, con la insistencia de que cada vez se aplica una regla, por lo menos una de las submetas utiliza un valor que ha sido descubierto en el ciclo anterior.

Es necesario describir algunos ejemplos de la operación $EVAL(p,R_1,\ldots,R_n)$ que se mencionó en el capítulo 5. Esta operación toma las relaciones $R_1,\ldots,R_n$ para las submetas de todas las reglas para el predicado $p$ y produce la relación para $p$ que se obtiene aplicando las reglas a la relación. Esto quiere decir que se buscan sustituciones para las variables de las reglas que convierten a cada submeta en una tupla de la relación correspondiente. El conjunto de sustituciones exitosas para las variables es llamada "relación para la tupla" y sus atributos son las variables que aparecen en la regla. Entonces, se toma cada tupla en la relación para la regla, se evalúa la cabeza con esa sustitución y se toma el átomo fundamental resultante como una tupla de la relación para $p$.

**Ejemplo 7.4:** en el ejemplo 7.2 se discutió la definición lógica de la adición. Este ejemplo es particularmente simple ya que no existen funciones de símbolos en las submetas. Así EVAL se opera casi de la misma manera que en el capitulo 5, pero cuando se encuentra una sustitución exitosa para las variables, se hace la sustitución en la cabeza y se obtienen términos no triviales como argumentos.

Al aplicar la evaluación de ecuaciones a las reglas (1) y (2) del ejemplo 7.2, se define una relación unaria *int*. Nótese que no existe un EDB para las reglas de este ejemplo.

Se inicializa la relación IDB para *int* a 0. Entonces, en el paso 1 del algoritmo 5.3, la regla (2) no produce resultados, porque *int(X)* no contiene tuplas. Sin embargo, la regla (1) produce la tupla 0 para la relación *int*.

En el paso 2, la primera regla produce 0 nuevamente, pero ahora la segunda regla tiene un valor, 0, para $X$ y, por lo tanto, produce la tupla *sig(0)* para *int*. Nótese que esta tupla tiene un componente y ese componente es un término con la función de símbolos *sig*.

En el tercer paso, $X$ también puede tomar el valor de *sig(0)*; técnicamente, la relación para la regla (2) es una relación sobre el atributo $X$, con el nombre {0,*sig(0)*}. Este nuevo valor para $X$, produce la nueva tupla *sig(sig(0))* para *int*. En general el i_esimo paso produce una nueva tupla, $sig^{i-1}(0)$ para *int*. Como un resultado el valor de la relación *int* en el punto fijo mínimo es el conjunto de los enteros no negativos {0,1,2,...}, con el entero $i$ representado por el término $sig^i(0)$. Así las reglas del ejemplo 7.2 definen a *int* como una relación infinita y un número infinito de iteraciones del ciclo de la evaluación semi ascendente es necesaria para obtener esta solución.

Un proceso similar ocurre cuando se siguen las reglas (3) y (4) para procesar la relación *sum*. En el primer paso, las submetas en los cuerpos de ambas reglas tienen relaciones vacías, así que no se producen tuplas para *sum*. En el paso 2, *int* contiene *0* así que la regla (3) produce (0,0,0) para *sum*.

En el paso tres *int* también contiene *sig(0)*, así que la regla tres produce

(sig(0),0,sig(0))

También *sum* contiene (0,0,0) así que la regla (4) produce (0,sig(0),sig(0)). En general se puede chequear que

$sum(sig^i(0), sig^j(0), sig^{i+j}(0))$

se produce en el paso *i+j+2*, usando la regla (3) si $j = 0$ y la regla (4) si $j > 0$. No se agrega otra tupla a *sum*. Así, *sum* también es una relación infinita en el punto fijo mínimo, requiriendo un número infinito de pasos para obtenerse.

Las relaciones infinitas como *int* y *sum* del ejemplo 7.4 no son computables ya que nunca se puede alcanzar el punto fijo mínimo de sus reglas definidas, no importando cuantas veces se corra la evaluación de ecuaciones. Por otro lado, la no computabilidad de las relaciones, no contradice el hecho de que estén bien definidas por el proceso de tomar el punto fijo mínimo, además es posible usar las reglas que definen una relación infinita para obtener algunas reglas en tiempo finito. En particular se puede contestar la pregunta de que si una tupla en particular se encuentra en tal relación, aunque nunca se pueda construir la relación completa.

**Ejemplo 7.5:** ahora reconsidérese el ejemplo 7.3. Para iniciar supóngase que están disponibles únicamente las reglas (1) y (2)

```
(1)   verdadero(g(S,T)) :- ir(S,T)
(2)   verdadero(g(S,T)) :- verdadero(g(S,U)) & verdadero(g(U,T)
```

En el primer paso, la relación *"verdadero"* se encuentra vacía, así que la regla (2) no produce tuplas. La relación EDB *IR* consiste en diez tuplas dadas en el ejemplo 7.3. Así , en el paso 1 la relación para la regla (1) es *IR(S,T)*. De cada una de las diez tuplas en esta relación, se construye un término *g(S,T)* y este término viene a ser el componente único de la relación unaria para *verdadero* . Así, después del paso 1, la relación para verdadero contiene *g(a,e)*, *g(e,d)* y otras ocho tuplas, cada una con un componente simple.

Subsecuentemente no se producen tuplas por la regla (1), pero ahora la regla (2) inicia a producir nuevas tuplas para *verdadero*. Para observar cómo, primero se evalúa la relación para la regla (2), al dar esa relación para *verdadero* se tienen las diez tuplas. Cada una de un componente, se muestra en la figura 7.3, hay que recordar que esa relación para esa regla tendrá los atributos *S*, *T* y *U* correspondientes a las tres variables que aparecen en el cuerpo de la regla, cada tupla debe tener los valores *s*, *t* y *u* para esas variables, así que las submetas *verdadero (g(S,U))* y *verdadero (g(U,T))* son tuplas de la relación para verdadero cuando se sustituye *s* para *S*, *t* para *T*, y *u* para *U*.

| |
|---|
| g(a,e) g(b,c) g(b,d) g(c,d) g(c,d)<br>g(d,b) g(d,c) g(d,e) g(e,a) g(e,d) |

**Figura 7.3 relación inicial para el predicado *verdadero***

Para ver qué valores de *s*, *t* y *u* se deben tener, se iguala cada submeta con cada tupla de la relación *verdadero*, que da los valores para las variables que aparecen en esa submeta, *S* y *U* para la primera submeta *U* y *T* para la segunda. Entonces, se toma la unión natural de este conjunto de tuplas para cada submeta, en este ejemplo particular, se debe tomar la submeta *verdadero (g(S,U))* contra cada una de las diez tuplas en la relación *verdadero* de la figura 7.3, por ejemplo, al comparar el componente *g(S,U)*, con la primer tupla, *g(a,e)*, se observa que se encajan y que los valores para *s*, *u* son *a* y *e*, respectivamente. Para cada una de las diez tuplas en *verdadero* se obtiene la relación:

{ae, bc, bd, cd, db, dc, de, ea, ed}

que se llamará $Q_1(S,U)$, para la primera submeta de la regla (2). Para la segunda submeta, *verdadero* $(g(U,T))$, también se construye una relación, que se llamará $Q_2(U,T)$, los valores de $Q_2$ se obtienen de la misma manera que $Q_1$ . Sin embargo, $Q_1$ es una relación sobre los atributos $S$ y $U$ y $Q_2$ sobre $U$ y $T$.

Ahora se toma la unión de $Q_1$ y $Q_2$ , en $R$ para obtener, la relación para la regla (2), $R$ contiene 22 tuplas, cada una con componentes para los atributos $S$, $V$ y $T$ respectivamente. Algunas de estas tuplas son *aea* [ de *ae* y *ea*] *bcd* [de *bc* y *cd*], *bde* y *edc*.

Para cada tupla, se construye la tupla de un componente $g(S,T)$ que la cabeza de la regla (2) indica que va en la relación para *verdadero*, por ejemplo, las cuatro tuplas pertenecientes $R$ mencionadas anteriormente, producen las siguientes tuplas para *verdadero* g(a,a), g(b,d), g(b,e) y g(e,c) de estas, sólo en la segunda existían en *verdadero*.

El proceso iterativo para procesar *verdadero* usando las reglas (1), (2) eventualmente, convergen, ya que no se construyen progresivamente términos más largos, aquí todos los términos para los componentes de la relación *verdadero* son de la forma g(x,y) donde $X$ y $Y$ son valores que aparecen en la relación *EDB IR* (estos valores son de *a* a *e* en este ejemplo). Si *IR* es finito, como debe ser, entonces, *verdadero* no puede generar un número infinito de tuplas diferentes.

Por otro lado, si se usan las reglas (5) a (7) del ejemplo 7.3 entonces se puede construir una infinidad de diferentes tuplas en *verdadero* por ejemplo, supóngase si se aplica la regla (7):

verdadero (and(X,Y)):- verdadero (X) & verdadero (Y)

En el paso 2, cuando *verdadero* tiene las diez tuplas de la figura 7.3 cualquiera de todas puede encajar con $X$ y con $Y$. Así, la relación para la primera submeta, *verdadero (x)* contiene los diez valores para $X$: {g(a,e), g(b,c),....) y la relación para la segunda submeta tiene la misma relación, pero el atributo es $Y$. Esto es $X$ y $Y$ siendo variables, pueden encajar como términos.

La relación para la regla (7) consiste de 100 tuplas sobre los atributos $X$ y $Y$, el producto cartesiano de la relación de la figura 7.3 con ella misma, cuando se sustituye para $X$ y $Y$ en la cabeza de la regla (7), se obtienen 100 tuplas para *verdadero* de las cuales *and(g(a,e), g(b,c))* es un ejemplo. Estas tuplas están disponibles en el próximo paso para relacionar $X$ y $Y$ en la regla (7) para construir términos más grandes como:

*and(and(g(a,e), g(b,c)), and(g(b,d), g(c,d)))*

y así sucesivamente.

Las reglas (5) y (6) presentan un problema más grande ya que no son reglas seguras. Para resolver el problema hay que reescribir las reglas para que sean seguras o no tratar de usarlas con el algoritmo de evaluación semi ascendente, donde se procesan todos los valores de los encabezados que vienen de las sustituciones de los términos fundamentales para las variables del cuerpo.

### 7.1.1 Igualación de términos.

Como se vio en los ejemplos anteriores, la operación que puede llamar igualando términos juega un rol importante en la generalización de la evaluación semi ascendente. Esto quiere decir que si se da una submeta, que sea una fórmula atómica normalmente conteniendo variables y se tarta de igualar con un átomo fundamental, es decir, con una tupla de la relación para el predicado que aparece en la submeta. Cuando se iguala una fórmula atómica arbitraria con un átomo fundamental, se sustituye para cada una de las variables de la primera fórmula para hacerla idéntica a la segunda. El siguiente algoritmo prueba cuando tal sustitución existe y la encuentra en tal caso.

**Algoritmo 7.1** Igualación de términos.

**Entrada:** una fórmula $F$ con las variables $X_1,\ldots,X_n$ (n>=0) y un átomo fundamental $G$.

**Salida:** si ésta existe, la única sustitución t sobre las variables $X_1,\ldots,X_n$ tal que cuando para toda $i$, se sustituye $t(X_i)$ para cada ocurrencia de $X_i$ en $F$, el resultado es $G$. Si no existe $t$ se muestra un error.

**Método:** inicialmente $t(X_i)$, no está definida para toda $i$. Entonces se aplica el proceso recursivo *igualar* que se muestra en el ejemplo 7.4 para $F$ y $G$. Si la llamada *igualar(F,G)* exitosamente responde verdadero, entonces $t(X_i)$ estará definida para toda $i$ y esta sustitución es la igualación de términos que se produce como salida.

La idea en el procedimiento *igualar(A,B)* es que si $A$ es una variable simple se debe tener $t(A)=B$, no importando cuál sea el término $B$. Si $A$ es un término o la fórmula atómica completa, entonces, se considera cada uno de los argumentos de $A$ y se comparan con los argumentos correspondientes de $B$, por la llamada recursiva de *igualar*, si el número de argumentos de $A$ y $B$ son diferentes, entonces, se da un error. Y se debe esperar a que todas las llamadas sean exitosas, para indicar una salida exitosa.

**Ejemplo 7.6:** sea *p(f(X,Y),g(X))* una fórmula atómica y el átomo fundamental

*p(f(h(a),b),g(h(a)))*

cada uno tiene dos argumentos, así que para igualarlos, se debe tomar el argumento *f(X,Y)* contra *f(h(a),b)* y *g(X)* contra *g(h(a))*, ya que *p* es su principal función.

Ahora se encuentra su principal función, *f*, el número de argumentos es dos en ambos casos, así que recursivamente se iguala *X* contra *h(a)* y *Y* contra *b*. Estas igualaciones son exitosas, dando $t(X)= h(a)$ y $t(Y)= b$. Entonces, se considera el segundo, donde las principales funciones son *g* y cada uno tiene un argumento *X* y *h(a)* respectivamente. Ahora, ya que *t(X)* esta definida, se debe verificar que t(X) es idéntica a la expresión contra la que *X* está siendo igualada. En este caso, se han chequeado todas las subexpresiones y la igualación es exitosa.

Un ejemplo simple donde la igualación produce un error, es *p(X,X)* contra *p(a,b)*. La primera llamada a *igualar*, genera la llamada a *igualar(X,a)* e *igualar(X,b)*. La primera es exitosa, pero, hace la asignación $t(X) = a$.

Entonces, la segunda llamada chequea que $t(X) = b$, encuentra que no está bien y muestra un error, causando que todo el algoritmo muestre un error.

```
Procedimiento igualar(A,B): booleana;
/* A es una expresión; B es una expresión sin variables
   responde verdadero si encuentra una igualación
   responde falso si no la encuentra.*/

si A es una variable simple entonces
      si t(A) esta definida entonces
            t(A) := B; Regresar(verdadero)
     sino
            si t(A) = B entonces Regresar (verdadero)
          sino  Retornar (falso)
sino
        sea A = Q(T1,...,Tk);
      /* aquí Q puede ser un predicado del la función de
         símbolos */
        si B no es de la forma Q(S1,...,Sk) entonces
            Regresar(falso)
        para i = 1 a k hacer igualar(Ti,Si)
        si alguna de las llamadas regresa falso entonces
           Regresar (falso)
        sino
           Regresar (verdadero)
fin.
```

**Figura 7.4 Algoritmo de igualación de términos.**

## 7.2 Procesamiento descendente de la lógica.

Se ha definido el significado de las reglas de la lógica para el punto fijo mínimo, iniciando solamente con los valores de EDB y repetidamente aplicando las reglas de derecha a izquierda, es decir del cuerpo a la cabeza. Este tipo de procesamiento es algunas veces llamado ascendente, porque implícitamente se construyen árboles que describen la prueba de una tupla IDB, trabajando hacia la raíz. Ese árbol tiene un nodo para cada tupla usada y los hijos de los nodos para la tupla m son las tuplas de las relaciones para cada una de las submetas que son usadas para construir m aplicando alguna regla. Técnicamente, la prueba del árbol debería ser un árbol acíclico ya que una tupla puede ser usada en varios lugares.

**Ejemplo 7.6:** considérense las reglas del ejemplo 7.3, la figura 7.5 muestra la prueba del árbol para el valor *verdadero(g(a,d))*, este valor fue construido por la regla (2) de los valores *verdadero(g(a,e))* y *verdadero(g(e,d))*; estos tomados de las relaciones para la primera y segunda submeta de la regla (2) respectivamente. Fueron colocadas en esa relación por la regla (1) ya que *ir(a,e)* e *ir(e,d)* son valores EDB dados. Nótese que el orden en el que las tuplas están colocadas en sus respectivas relaciones de abajo hacia arriba de acuerdo con el árbol de la figura 7.5. Las salidas *ir(a,e)* e *ir(e,d)* están en sus relaciones EDB inicialmente. Entonces, los átomos del segundo nivel del árbol *verdadero(g(a,e))* y *verdadero(g(e,d))* se infieren en el paso 1, mientras que la raíz se infiere en el paso dos.

El procesamiento ascendente de la lógica es algunas veces llamado *encadenamiento hacia atrás*. La explicación es que las reglas, algunas veces, son regeneradas, así como tienen una dirección natural en las que deben ser

aplicadas, es decir, del cuerpo a la cabeza o del antecedente al consecuente: "si tal y tal son verdaderas, entonces, esto también es verdadero". Si se razona de esta manera se puede decir que es un razonamiento hacia adelante.

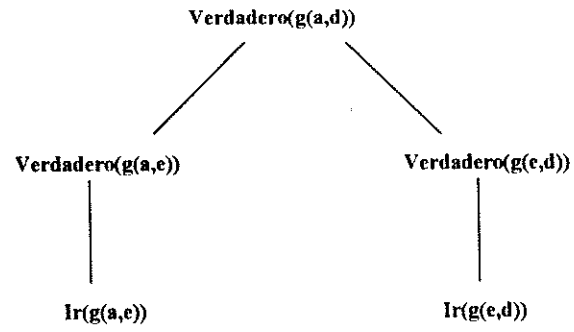



**Figura 7.5 Prueba del árbol**

### 7.2.1 Razonamiento desde las metas.

Existe otro modo de razonamiento, llamado de arriba hacia abajo o encadenamiento hacia atrás, donde se inicia con la meta que se quiere probar; se consideran las formas en las que se puede probar exitosamente la meta.

**Ejemplo 7.7:** supóngase que se dan las reglas del ejemplo 7.3 y la meta *verdadero(g(a,d))* para probar. Una forma de hacerlo es aplicando la regla (1) del ejemplo 7.3:

*(1) verdadero(g(S,T)) :- ir(S,T).*

De tal forma que la cabeza se convierte en la meta deseada. Para deducir qué tupla debe tener para estar en la relación para la regla (1), se hace una igualación de términos de la cabeza, *verdadero(g(S,T))*, contra la meta, verdadero(g(a,d)), encontrando que *S=a* y que *T=d*. Así, al aplicar la regla (1) de tal forma que la tupla sea *verdadero(g(a,d))*, se necesita una tupla *(a,d)* en la relación para la regla (1); hay que recordar que la relación tiene dos componentes, correspondientes a las variables *S* y *T*, en ese orden. La única forma de obtener *S=a* y *T=d* es: si la submeta *ir(a,d)* es una tupla de la relación *IR*. Pero *IR* es una relación EDB y se supone que consiste de las diez tuplas listadas en el ejemplo 7.3, estas tuplas no incluyen (a,d), así que la regla (1) no ayuda para llevar a cabo la meta *verdadero(g(a,d))*.

Ahora usando la regla (2) del ejemplo 7.3

(2) verdadero(g(S,T)) :- verdadero(g(S,U) & verdadero(g(U,T)).

para probar la meta inicial, *verdadero(g(a,d))*. La cabeza de la regla (2) es la misma que la cabeza de la regla (1), así que se iguala, nuevamente, *S=a* y *t=d* si se hacen estas sustituciones en el cuerpo de la regla (2) se obtienen las submetas *verdadero(g(a,U))* y *verdadero(g(U,d))*.

Se trabaja con cada una de estas, primero tratando con la regla (1) y luego con la regla (2). Cuando se iguala la cabeza de la regla (1) contra *verdadero(g(a,U))*, se encuentra algo que no se ha visto con anterioridad: ambas fórmulas tienen variables. La meta está pidiendo algún valor para *U* en donde

*verdadero(g(a,U))* pueda ser probado. En este caso la forma de igualar es obvia; *S* es la cabeza en la regla (1), se iguala a *a* y *T* se iguala a *U*. Si se sustituye *a* por *S* y *U* por *T*, el cuerpo de la regla (1) se convierte en *ir(a,U)*, ya que *ir* es un predicado EDB, solamente se puede satisfacer la meta encontrando un valor de *U* tal que *(a,U)* es una tupla de *IR*. En este caso existe exactamente un valor que *U* puede tomar, llamado *e*, aunque en general podría ser cero, uno o más valores.

El encadenamiento de *U* al valor e, que ocurre cuando se considera la primera submeta de la regla (2), afecta la segunda submeta, que es *verdadero(g(U,d))*, pero ahora se convierte en *verdadero(g(e,d))*. Se puede satisfacer, primero por la regla (1). Esta tarea es fácil ya que el cuerpo de la regla (1) se convierte en *ir(e,d)* e *ir(e,d)* es una tupla de la relación EDB *IR*. Ahora se han satisfecho todas las submetas de la regla (2) y la meta original ha sido probada. Su prueba es el árbol de la figura 7.5.

### 7.2.2 Seguimiento hacia atrás.

En el ejemplo 7.7, se procedió directamente de la meta a la prueba. El único desvío que se tomó fue el intento inicial de probar *verdadero(g(a,d))* por la regla(1). Sin embargo, hay otras rutas que se pudieron tomar, sin éxito alguno. Por ejemplo, si *ir(e,d)* no fuera un valor de la relación EDB, entonces la submeta *verdadero(g(e,d))* también fallaría. Se debe de regresar a la primera submeta de la regla (2) y tratar de encontrar otro valor para U en lugar de e.

Ya que no existen más tuplas de la forma *(a,U)* en *IR*, se trata de alcanzar la meta *verdadero(g(a,U))* aplicando la regla(2). Antes de hacer esto, se debe entender que el uso de *U* en la meta y en la regla (2) no es el mismo. Ya que *U* en la regla (2) es una variable local con el mismo nombre mientras que es una variable en la meta pero no igual. Así para evitar confusión se debe reescribir la regla(2), para la aplicación de la submeta *verdadero(g(a,U))* con una nueva variable en lugar de U, por ejemplo:

```
verdadero(g(S,T)) :- verdadero(g(S,V)) & verdadero(g(V,T)).
```

Cuando se iguala, técnicamente unificar, la cabeza de esta regla con la meta *verdadero(g(a,U))* se encuentra que *S=a* y *T=U*. Se está alcanzando la meta *verdadero(g(a,V))* que realmente no es diferente de la meta *verdadero(g(a,U))* que es lo que se trata de alcanzar.

Esta exploración ilustra el punto de que la búsqueda descendente con seguimiento hacia atrás tratando las reglas alternativas, cuando una falla, puede fácilmente convertirse en un ciclo. En efecto los ciclos pueden ocurrir cuando se procesa una búsqueda descendente, aunque las reglas sean de Datalog y, por lo, tanto la solución ascendente siempre debe converger. Por ejemplo las reglas de la cerradura transitiva de un grafo

```
camino(X,Y) :- trazo(X,Y).
camino(X,Y) :- camino(X,Z) & camino(Z,Y)
```

no son realmente diferentes de las reglas (1) y (2) del ejemplo 7.3. Cuando se inicia con una meta como *camino(a,b)*, donde para ningún valor de *W*, (a,W) es una tupla de la relación EDB para *trazo*, se encamina al mismo ciclo que se describió anteriormente, generando las metas *camino(a,Z)*, ..., *camino(a,$Z_n$)*.

## 7.3 Unificación.

La unificación es el paso principal cuando se procesan las reglas de la lógica de una manera descendente. Para esta operación se toman dos fórmulas atómicas, cada una con variables y encuentra una sustitución para cada variable, así que las fórmulas vienen a ser idénticas. Tal sustitución es llamada *unificador*. Al contrario de la igualación de términos, en la unificación, las expresiones que se sustituyen para las variables, algunas veces ellas mismas involucran variables. Así que cualquier expresión puede ser sustituida por estas variables y las fórmulas resultantes seguirán siendo idénticas. Como una consecuencia, puede haber más de un unificador para un par de fórmulas.

Se muestra un algoritmo que produce el unificador más general de dos fórmulas. La sustitución tiene la propiedad que es un unificador y cualquier otro unificador puede ser construido del unificador más general por la sustitución de sus variables. Así que el unificador más general es la sustitución más simple que hace que dos fórmulas sean idénticas.

En el siguiente algoritmo se asume tácticamente, que la primera fórmula atómica es una meta y que la segunda es la cabeza de una regla que se espera se unifique con la meta y, por lo tanto, puede ayudar a deducir la meta. Este punto de vista tiene dos consecuencias.

1. Ya que las variables de las reglas son locales, se debe asumir que no existen variables en común entre las dos fórmulas.

2. Si se tiene la opción de sustituir una variable de la primera fórmula por una variable de la segunda o viceversa, se sustituye por la variable de la segunda. Esta opción se hace más transparente cuando las variables de una meta se acarrean hacia atrás para sus submetas, pero, esto no es esencial para la aproximación descendente.

**Algoritmo 7.2** Proceso del unificador más general.

**Entrada:** las fórmulas atómicas *F* y *G* con un conjunto distinto de variables.

**Salida:** el unificador más general de *F* y *G* o un indicador de que no existe.

**Método:** existen dos fases. En la primera se determina cuales de las subexpresiones de las fórmulas *F* y *G* deben ser idénticas en el unificador. En la segunda fase se determina donde existe un unificador. Se chequea para una contradicción inmediata donde dos subexpresiones idénticas tienen dos diferentes símbolos no variables que deben ser igualados y si se encuentra ese par, se concluye que no existe el unificador. Entonces, se procede a construir el unificador, que debe existir, en principio, pero puede ser infinito, por lo tanto, no se puede usar y se dice que el unificador más general no existe.

Fase I. Encontrando subexpresiones equivalentes. El primer paso es construir para *F* y *G* representando sus estructuras como expresiones. Esto es, se define el árbol para el término o fórmula atómica *T*, recursivamente, como sigue:

1. Si *T* es una variable o una constante entonces el árbol para *T* es un nodo simple etiquetado *T*.

2. Si $T$ es de la forma $Q(T_1,\ldots,T_k)$ donde Q es un predicado o una función de símbolos y $T_1,\ldots,T_k$ son términos, entonces, el árbol para $T$ tiene una raíz etiquetada $Q$ y tiene hijos que son raíces de los árboles para $T_1,\ldots,T_k$, en ese orden, desde la izquierda.

Ahora que ya se tienen construidos los árboles para $F$ y $G$ se deben de agrupar sus nodos en clases equivalentes, representadas por la relación equivalente $\circ$ , si no se tiene éxito al completar la construcción de $\circ$ por las reglas de abajo, entonces, no existe el unificador. Intuitivamente, $n \circ m$ si y solo si, en algún unificador las expresiones representadas por los subarboles con raíces $n$ y $m$ deben ser idénticas. las reglas que definen $\circ$ son:

a) si $r_F$ y $r_G$ son las raíces de los dos árboles, entonces, $r_F \circ r_G$. Esta regla se sigue de la definición de unificación; se necesita que las dos fórmulas atómicas sean la misma;

b) si $n \circ m$ entonces los nodos $m$ y $n$ deben tener el mismo número de hijos; sino, no existe un unificador para $F$ y $G$. Si tiene hijos $c_1,\ldots,c_k$ y $d_1,\ldots,d_k$, respectivamente, entonces $c_i \circ d_i$ para toda $i$. El motivo para esta regla es que en ese orden para las expresiones de $n$ y $m$ sean iguales, deben tener el mismo número de subexpresiones, y las subexpresiones correspondientes deben ser idénticas;

c) si $n$ y $m$ están etiquetadas por la misma variable, entonces $n \circ m$. La razón es que se debe tomar sólo una sustitución para la variable $X$, así que cada nodo etiquetado $X$ debe ser la misma expresión dentro de cualquier unificador;

d) $n \circ n$ para cualquier nodo $n$; si $n \circ m$ entonces $m \circ n$, y si $n \circ m$ y $m \circ p$, entonces $n \circ p$

Fase II. Buscando y construyendo el unificador más general. Ya agrupados los nodos en clases equivalentes, se considera cada clase de equivalencia en turno. Primero si cualquier clase tiene dos nodos que están etiquetados por dos símbolos diferentes y ninguno es una variable, entonces no existe un unificador. La razón es que no se puede hacer sustitución para variables para que dos subexpresiones sean idénticas si sus raíces son diferentes símbolos y no es una variable.

Asumiéndose que todavía es posible un unificador, se trata de encontrar un orden para tratar la clase de equivalencias y encontrar para cada clase una expresión representativa. Se usa t para el unificador. Inicialmente, sea t indefinida para cada variable. Las siguientes reglas definen t para algunas de las variables; si existe algún unificador entonces eventualmente definirá todas las variables. La primera regla sirve como una base y la segunda puede ser aplicada repetidamente.

1. Si una clase contiene sólo nodos etiquetados, por variables, tomar una, prefiriendo una variable de la primera expresión si existe una. Sea la variable seleccionada igual a $X$. Entonces para cada variable $Y$ etiquetando un nodo en la clase, sea $t(Y) = X$. Nótese que $t(X) = X$ como consecuencia.

2. Supóngase una clase contiene uno o más nodos etiquetados por variables también por lo menos un nodo etiquetado por una constante, símbolo de funciones o predicado. Supóngase también que uno de los últimos nodos representa una expresión $E$ en donde todas las variables, si hay, tienen definido t para ellas. Sea t(E) el resultado de sustituir $t(X)$ para todas

las ocurrencias de *X* en *E*, para cada variable *X*. Entonces, para cada variable *Y* etiquetando un nodo en esta clase sea t(*Y*) = t(*E*).

Si en los pasos anteriores se tiene éxito al definir t(*X*), para todas las variables *X*, entonces, t es el unificador más general, de otra forma, no existe el unificador y esto quiere decir que las fórmulas son no unificables.

**Ejemplo 7.13:** considérese un caso bastante complicado donde el unificador más general existe. Las dos fórmulas atómicas son:

*p(f(X,g(X)),g(g(Y)))*

*y*

p(f(g(U),V),g(V))

los árboles que representan estas expresiones se muestran en la figura 7.6. Los nodos están numerados de 1 a 15 para referencia.

se agrupan los nodos en clases de equivalencia. La regla (a) indica que 1°9, porque estas son las raíces de los árboles de las dos fórmulas. La regla (c) indica que 4 ° 7 porque ambas están etiquetadas con la variable *X*, y 13 ° 14 porque están etiquetadas como *V*. La regla (b) indica que los hijos correspondientes del 1 al 9 son equivalentes; esto es 2 ° 10 y 3°11. Asimismo, los hijos correspondientes al par equivalente 2 y 10 son equivalentes, así que 4 ° 12 y 5 ° 13.

Ahora se usa la regla (d) para inferir que 4, 7 y 12 todos pertenecen a una clase de equivalencia, es decir que cada uno es equivalente a los otros dos. La razón es que 4 ° 7 y 4 ° 12 ambos han sido encontrados. Similarmente, 13 ° 14 y 5 ° 13 indican que 13, 14 y 5 son todos equivalentes.

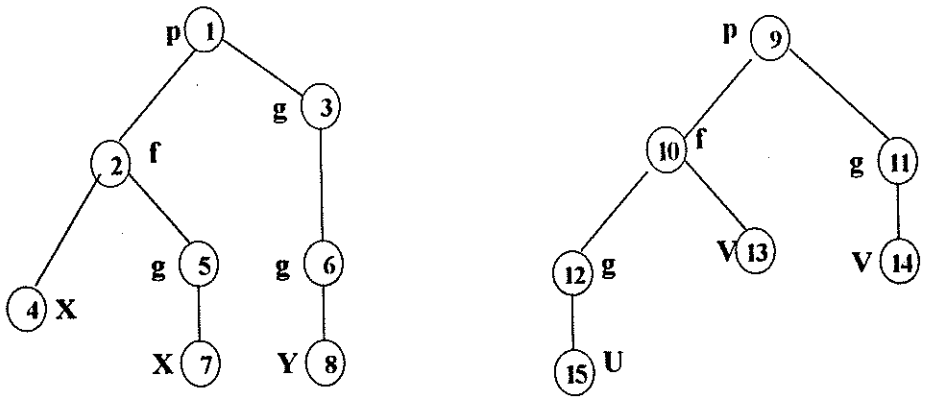



**Figura 7.6. Arboles de expresión para las fórmulas atómicas del ejemplo 7.8.**

Ya que 3 ° 11, la regla (b) indica que sus hijos son también equivalentes, esto es, 6 ° 14. Así por (d) 6 también es equivalente a 5 y 14.

Ahora, ya que 5 ° 6, la regla (b) indica que sus hijos son equivalentes, así que 7 ° 8. Así 8 se une a la clase de 4, 7, y 12, y las clases equivalentes son:

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

Ahora, no se pueden encontrar más pares equivalentes por las reglas (a) a (d). Un chequeo honesto indica que no existe una clase de equivalencia que contenga dos símbolos no variables distintos. Por ejemplo los nodos de la clase {5,6,13,14} están todos etiquetados con $g$ o $V$ y de éstas, sólo $g$ es una no variable.

Ahora se trata de definir la sustitución t. La clase que contiene a 15 solamente contiene variables, se toma una, la única opción es $U$ y se define $t(U)=U$. Ahora se considera la clase 4 ° 7° 8 ° 12, donde sólo 12 no está etiquetado como $X$ o $Y$ y el nodo 12 representa la expresión $g(U)$, cuya única variable tiene definida t. Por lo tanto se hace $t(X)$ y $t(Y)$ iguales a

$$t(g(X)) = g(t(U)) = g(U)$$

Ahora se considera la clase 5 ° 6 ° 13 ° 14, dos de los nodos, 13 y 14, están etiquetados como $V$, mientras que los otros representan las expresiones $g(X)$ y $g(Y)$. Ambas expresiones tienen t definida para sus variables y cualquiera que se tome, se obtiene $t(V) = g(g(U))$, ya que $t(X) = t(Y) = g(U)$. Se ha completado ahora la definición del unificador más general

$t(U) = U$ $\qquad$ $t(X) = g(U)$
$t(U) = g(U)$ $\qquad$ $t(V) = g(g(U))$

se obtiene la fórmula atómica

$$p(f(g(U),g(g(U))),\ g(g(g(U))))$$

cuando se hace esta sustitución en cualquiera de las fórmulas atómicas dadas.

**Ejemplo 7.9:** considérese un par de fórmulas que no tienen un unificador. Considérense $p(X,X)$ y $p(f(Y),g(Z))$. Sus árboles se muestran en la figura 7.7, y, las clases de equivalencias obtenidas aplicando las reglas (a) a la (d) del algoritmo 7.2 son:

7 $\qquad$ 8 $\qquad$ 1 ° 4 $\qquad$ 2 ° 3 ° 5 ° 6

Ya que 5 ° 6, pero 5 y 6 tienen diferente función de símbolos como etiquetas, no hay manera de construir un unificador. Intuitivamente estas expresiones no pueden ser unificadas ya que no existe alguna forma de que $X$ se convierta en $f$ de algo, ni $g$ de algo al mismo tiempo. Debe quedar claro que solamente las variables son sujeto de sustitución, así que no se puede sustituir $f$ por $g$.

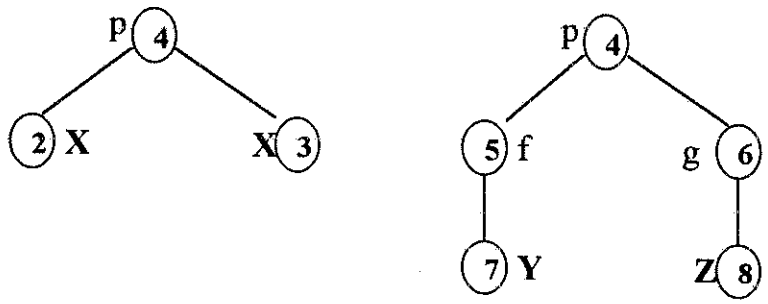



**Figura 7.7 árboles de expresión para fórmulas no unificables.**

**7.4 Arboles de reglas y metas.**

Cuando se exploran las metas, sus reglas y sus submetas, se debe considerar cada meta o regla como un evento. Se puede construir un árbol de eventos, llamado *árbol de reglas y metas*, donde cada evento está representado por un nodo. Asociado con cada nodo hay una descripción de la meta o regla, con variables nombradas. El árbol de reglas y metas para una meta dada, $G_0$, se construye de la siguiente manera:

1. La raíz es una meta para $G_0$.

2. Supóngase que el nodo $n$ es un nodo meta para $G$, y sea $p$ un predicado de $G$. Entonces para cada regla $r$ de $p$, unificar la cabeza de $r$ con $G$. Si la unificación es exitosa, crear un nodo para la versión modificada de la regla $r$ que es un hijo del nodo $n$. Específicamente, se ejecutan los siguientes pasos para obtener la regla modificada $r$.

   a) Primero, renombrar las variables de $r$, si es necesario, así que $r$ no tiene variables en común con $G$.

   b) Cuando se ejecuta la unificación de la cabeza de $r$ con $G$, cuando sea posible, sustituir las variables que aparecen en la cabeza de $r$.

   c) Cuando se sustituye una variable $X$ que aparece en la cabeza de $r$, se debe hacer esa sustitución para todas las ocurrencias de $X$, no sólo en la cabeza de $r$.

   d) Todas la variables $Y$ de $r$ que no aparecen en la cabeza son locales para $r$. Se identifican esas variables por la activación particular del registro para $r$ subscribiéndose con el número de ancestros de este registro de activación que es un nodo regla, incluyendo el nodo presente.

3. Los hijos de un nodo regla son nodos metas correspondiendo a cada una de las submetas en el cuerpo de la regla. En estos nodos meta, como en la regla la sustitución de 2(c) ha sido realizada.

**Ejemplo 7.10**: considérense las siguientes reglas para definir los ancestros de una relación EDB *par*, donde *par(C,P)* significa que *P* es pariente de *C*.

$r_1$: anc(X,Y) :- par(X,Y).
$r_2$: anc(X,Y) :- par(X,Z) & anc(Z,Y).

$r_2$ indica que $Y$ es un ancestro de $X$ si existe algún individuo $Z$ tal que $Z$ es un pariente de $X$ y $Y$ y es ancestro de $Z$.

Supóngase que la relación $P$ para el predicado *par* es como se muestra en la figura 7.8, las aristas representas *pares(x,y)* en $P$. Se inicia con la meta *anc(j,W)*, que es, encontrar todos los ancestros $j$; esta meta se convierte en la raíz del árbol de metas y reglas, los niveles de éste se muestran en la figura 7.9.

Se crean hijos para la raíz de cada una de las reglas $r_1$ y $r_2$. Para $r_1$ se unifica la cabeza, *anc(X,Y)* con la meta *anc(j,W)* y se encuentra que $X \circ j$ y $Y \circ W$. En el unificador más general se debe usar la constante $j$ en lugar de la variable $X$, pero se puede usar la variable $W$ o $Y$. Ya que se desea eliminar las

variables de la cabeza, se usa *W* así que el unificador más general convierte la cabeza de $r_2$ en *anc(j,W)* y sucede que es la misma que la meta. La misma sustitución de *j* para *X* y *W* para *Y* se hace a través de la regla $r_1$, así que para la regla en el primer hijo de la raíz se tiene

*anc(j,W) :- par(j,W).*

Cuando se considera $r_2$, se tiene la misma cabeza, *anc(X,Y)*, que para $r_1$, así que se ejecuta la misma unificación de la meta con la cabeza, produciendo la misma sustitución de *j* para *X* y *W* para *Y*. Se aplica esta sustitución a la segunda regla y se crea la variable local $Z_1$ en lugar de *Z*; el subíndice *1* se usa porque se encuentra en el primer nivel de nodos reglas. Así para el segundo hijo de la raíz se tiene

*anc(j,W) :- para(j,*$Z_1$*) & anc(*$Z_1$*,W).*

para la regla $r_2$.

Ahora se considera el primer hijo de la raíz, el hijo correspondiente a $r_1$. Este nodo tiene un hijo para cada meta en el cuerpo de $r_1$. Como existe sólo una submeta, existe sólo un hijo, un nodo meta para *par(j,W)*. Como *par* es un predicado EDB y por lo tanto, no tiene reglas, este nodo meta es una hoja.

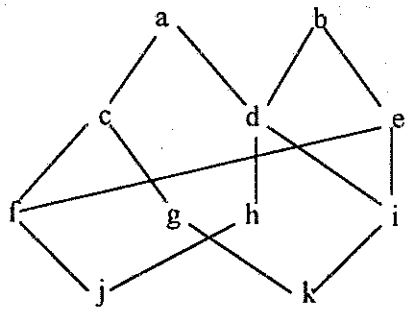



**Figura 7.8 La relación *P* para el predicado *par*.**

Ahora considérese el segundo hijo de la raíz, la que corresponde a $r_2$. Tiene dos nodos meta hijos, el primero para la submeta *par(j,*$Z_1$*)* y el segundo para *anc(*$Z_1$*,W)*. El primero tiene un predicado EDB, es una hoja. El segundo, tiene un nodo regla hijo para las reglas $r_1$ y $r_2$. Se unifican las cabezas de la dos reglas, que ambas son *anc(X,Y)*, con la submeta *anc(*$Z_1$*,W)*. Cuando se unifican las cabezas de $r_1$ y $r_2$ con esta submeta, se tiene las dos reglas en los nodos regla hijos del nodo meta *anc(*$Z_1$*,W)* se convierten en

anc($Z_1$,W) :- par($Z_1$,W).
anc($Z_1$,W) :- par($Z_1$,$Z_2$) & anc($Z_2$,W).

Nótese que la variable local *Z* se convierte en $Z_2$ en la segunda regla, porque se introduce en el segundo nivel de nodos reglas.

La figura 7.9 muestra la expansión del árbol de reglas y metas para los niveles que se han tomado y un nivel adicional de nodos metas. El proceso que se ha seguido se repite indefinidamente , con cada nodo meta ancestro que tiene hijos

para las dos reglas. La primera regla termina en una hoja *par*, pero la segunda regla produce una hoja *par* y otro nodo meta *anc*.

7.4.1 Propagación de cadenas y respuestas.

Así como se construye un árbol de reglas y metas, se construyen relaciones en cada nodo.

1. En cada nodo meta *n*, se tiene una relación de encadenamiento *M*. La meta en *n* tendrá cero o más argumentos que son limitados, en el sentido de que todas las variables que aparecen en esos argumentos son valores limitados dados, ya sean de una constante, como *j* en la figura 7.9, que se pasa de arriba, o por un conjunto finito de tuplas que se pasan de un lado porque aparecen en una submeta anterior. Las tuplas de *M* representan encadenamientos para esos argumentos limitados.

2. En cada regla nodo existe una relación suplementaria $S_0,\ldots,S_k$ con $S_i$ correspondiente al punto en el cuerpo de la regla con *i* submetas a la izquierda. Las relaciones suplementarias representan el encadenamiento de esas variables que han sido limitadas por el encadenamiento de la relación para los nodos padres de la meta, aplicado a la cabeza de las reglas o aparece en una submeta previa de la regla.

3. Ambos, los nodos de las reglas y de las metas, tienen relaciones de respuestas o resultados. En el caso de un nodo meta, esta relación es el conjunto de tuplas que son iguales a la meta y que son probablemente de las reglas, mientras que para un nodo regla, esta relación tiene aquellas tuplas que son iguales a la cabeza de la regla y que se pueden probar por una secuencia de aplicaciones de la regla, terminando con la regla en ese nodo. La relación resultante para la raíz nodo meta, es la respuesta de la búsqueda.

**Ejemplo 7.11:** considérese el procedimiento del árbol de la figura 7.9. Supóngase que se trabajó por un momento con el hijo izquierdo de la raíz. Para procesar la relación para ese nodo regla, se debe encontrar la relación para el nodo meta *par(j,W)*. Ya que *par* es un predicado EDB, se necesita solamente igualar los términos de la meta *par(j,W)* con las tuplas en la relación *P*; es decir que se encuentran aquellas tuplas con el primer componente *j*, y para cada tupla el segundo componente se convierte en un valor para *W*. Así la relación para el nodo meta *par(j,W)* es $s_{\$1=j}(P)=\{jf,jh\}$, de acuerdo a los datos para *P* dados en la figura 7.8. Nótese que el resultado de la relación es sobre todos los argumentos de *par*, aunque se sepa que el primer argumento debe ser *j*.

Parece que se pueden pasar simplemente estas dos tuplas al nodo regla que es el hijo izquierdo de la raíz y, entonces, a la raíz misma, para demostrar que *jf* y *jh* son tuplas que igualan a la raíz meta, *anc(j,W)*. Sin embargo, este ejemplo no tiene funciones de símbolos y las variables tienen nombres convenientes. Así que no se puede pensar que el proceso de pasar tuplas de las metas a las reglas y de las reglas a las metas, es simple en general.

El traslado que se debe ejecutar se sugiere por los siguientes pasos, que son similares a la evaluación ascendente:

1. convertir la relación {jf, jh} para *par(j,W)* en una relación sobre alguna de las variables que aparecen para la regla $r_1$,

```
anc(j,W) :- par(j,W).                    (I)
```

que aparece en el hijo izquierdo de la raíz en la figura 7.9. se iguala la submeta *par(j,W)* con cada tupla de la relación {jf,jh}, que se debe pensar como {*par(j,f)*,*par(j,h)*} obteniendo los valor de *f* y *h* para *W;*

2. tomar la unión natural de la relación para cada submeta de (I). Ya que solamente hay una submeta, la unión es trivial en este caso, y se obtiene la relación {f,h} para el cuerpo de (I). El esquema de la relación sigue siendo *W;*

3. procesar la relación para la cabeza, *anc(j,W)*, de (I). Esto se hace, convirtiendo las variables a argumentos para el cuerpo y cabeza de (I). Así, la cabeza de (I) da la relación {jf,jh}.

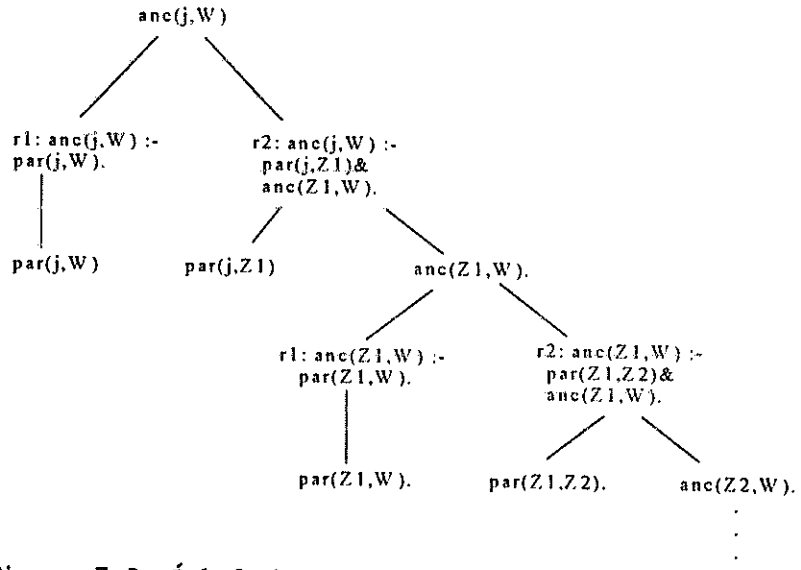



**Figura 7.9 Árbol de reglas y metas para la meta *anc(j,W)*.**

4. ahora la relación para la cabeza de (I), se pasa a su padre, el nodo raíz en este caso. Así *jf* y *jh* son respuestas de la búsqueda, es decir que *f* y h son ancestros de *j*.

Ahora, se considera el hijo derecho de la raíz, un nodo regla para la regla $r_2$

anc(j,W) :- par(j,$Z_1$) & anc($Z_1$,W). (II)

se inicia trabajando la primera submeta, *par(j,$Z_1$)*. Ya que esta meta tiene un predicado EDB, se puede resolver por observación y se encuentra una relación familiar para esta submeta, {*jf,jh*}. Estas tuplas se pasan al hijo derecho de la raíz donde establecen que la relación para la submeta *par(j,$Z_1$)* es {*f,h*}, pero, aquí tiene el esquema $Z_1$.

Ahora se pasa la información, para establecer que para la submeta *anc($Z_1$,W)*, $Z_1$ sólo puede tener los valores *f* y *h*. El primer argumento de *anc($Z_1$,W)* se convierte en limitada para la relación unaria {f,h}, así como el primer argumento de *par(j,W)* era limitado para *j*. La diferencia es que en el último caso, el encadenamiento fue establecido por unificación, como se creó el árbol de metas y reglas, el encadenamiento se establece por el paso de

información a través de las relaciones suplementarias , después de que se ha explorado parte del árbol.

Con este encadenamiento para $Z_1$, se puede explorar el árbol desde $anc(Z_1,W)$, y una secuencia de eventos permitirá determinar la exploración del árbol infinito, teóricamente. Se encuentra que cuando se toma el nodo meta $anc(Z_2,W)$, se ha establecido el encadenamiento {c,d,e} para $Z_2$; nótese que son los abuelos de $j$ en la figura 7.7, así como el encadenamiento para $Z_1$ son los padres de $j$. Entonces cuando se explora el nodo meta $anc(Z_3,W)$ se nota que $Z_3$ es limitado para los bisabuelos de $j$, esto es {a,b}. Después cuando se alcanza el nodo meta $anc(Z_4,W)$, $Z_4$ se limita a 0.

En este punto se sabe que no existe razón para explorar más. ya que la relación para $anc(Z_4,W)$, está vacía. Nótese cómo el paso de información ha establecido que no existen tuplas para la relación de la submeta $anc(Z_4,W)$, que puede producir respuestas a la búsqueda original. En efecto se ha usado una secuencia para descubrir que una de las submetas en el árbol de metas y reglas se encuentra vacía y, por lo tanto, esa submeta no necesita ser examinada. Sólo se necesita saber que la relación *par* es finita y no tiene ciclos y se garantiza que después de un número de niveles, alguna $Z_i$ se encadenará a 0.

Así se puede calcular después de una cantidad finita de tiempo que la relación para el nodo meta $anc(Z_1,W)$ es :

{fc, fe, hd, fa, fb, ha, hb} (III)

esta relación se traslada a una relación idéntica con el esquema $Z_1W$ para la submeta $anc(Z_1,W)$ de (II) para $r_2$ y esa relación se une con la relación {f,h}, que tiene el esquema $Z_1$, para la submeta $par(j,Z_1)$ en (II). El resultado es una relación con el esquema $Z_1W$ para el cuerpo de $r_2$ en el hijo derecho de la raíz, es idéntica a la relación (III) y a su esquema.

Finalmente, se construye la relación para la cabeza, $anc(j,W)$ de $r_2$, juntando la constante $j$ con el segundo componente de las tuplas en (III). Estas tuplas,

{jc, je, ja, jb, jd}

juntas con las tuplas *jf* y *jh* que se encontraron anteriormente, forman la relación completa para la raíz de la figura 7.9.

## 7.5 Procesó de las relaciones durante la expansión de los árboles de reglas y metas.

Se mostrará de forma general los algoritmos para el procesamiento de las relaciones asociadas con los nodos de un árbol de reglas y metas. Primero hay que formalizar el tipo de relaciones que se procesan:

1. en cada nodo meta existe una relación de respuesta o resultado, cuyos componentes corresponden a los argumentos de la meta;

2. en cada nodo regla existe una relación de respuesta o resultado cuyos componentes corresponden a los argumentos de la cabeza de la regla;

3. en cada nodo regla existe una relación de encadenamiento, cuyos componentes corresponden a los argumentos de la meta que son limitados. Los mismos argumentos son limitados en la cabeza para cada nodo regla hijo de



PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

este nodo meta. Se dice que las variables que aparecen en uno de esos argumentos limitados de la cabeza de una regla, son limitados por la cabeza de la regla;

4. en cada nodo regla para una regla con $k$ submetas existen relaciones suplementarias $S_0,\ldots,S_k$. Los argumentos de una relación suplementaria corresponden a alguna de las variables de la regla. En particular $S_i$ tiene argumentos que corresponden a las variables que son relevantes y limitadas después de considerar las primeras $i$ submetas. Hay que recordar que las variables son limitadas ya sea por aparecer en un argumento limitado de la cabeza o por aparecer en una de las primeras $i$ submetas; las variables son relevantes si aparecen ya sea en la cabeza o en las primeras $i+1$ o submetas subsecuentes.

Hay que entender que los encadenamientos son tuplas, con uno o más componentes. Aproximadamente, una tupla m encadena ciertos argumentos o variables que significa que durante el algoritmo de seguimiento hacia atrás, estos argumentos o variables tienen simultáneamente dados los valores encontrados en los componentes de m.

Cada relación de (1) a (4) es procesada de los valores de una o más relaciones. Esto requiere escoger una delicada secuencia de operaciones para asegurarse de que cada tupla corresponde a alguna relación en algún nodo del árbol de reglas y metas.

### 7.5.1 Paso de información.

Considérese un nodo regla con la regla $H$ :- $G_1$ & ... & $G_k$. Para $i$= 1,...$k$, se procesa $S_i$ de $S_{i-1}$ y la relación para la submeta $G_i$. Es decir, sea la relación resultante para el hijo de este nodo regla corresponde a la submeta $G_i$ igual a $R_i$, y sea $Q_i$ la función de pasar las variables a argumentos de $G_i$. Entonces, $S_i$ se procesa por la unión de $S_{i-1}$ |x| $Q_i$ y proyectando cualquier variable irrelevante.

Ejemplo 7.12: considérese la regla

p(f(X),Y) :- $q_1$(X,U,V) & $q_2$(G((U,V),f(U)) & $q_3$(W,X,Y)).

supóngase que se da una relación $S_0(X)$ que provee un conjunto de valores para la variable $X$. Las relaciones suplementarias son procesadas por:

$S_1$(X,U) :- $p_{XU}$($S_{0(X)}$ |x| $Q_1$(X,U,V)).
S2(X,W) :- $p_{XW}$($S_{1(X,U)}$ |x| $Q_2$(U,W)).
$S_3$(X,Y) :- $p_{XY}$($S_{2(X,W)}$ |x| $Q_3$(W,X,Y)).

donde $Q_i$ es la relación para la submeta $i$ trasladada desde el punto de vista de argumentos de $q_i$ a el punto de vista de variables.

Por ejemplo, $X$ y $U$ son atributos de $S_1$ porque ambos son limitados y relevantes. En particular, $X$ es limitado porque está limitado por la cabeza, mientras que es relevante porque es usado después en la submeta $q_3$ y en la cabeza. $U$ está limitado en $Q_1$ y aparece subsecuentemente en $q_2$. $V$ no es un atributo de $S_1$ porque aunque está limitada por $Q_1$ no aparece ni en $q_2$ ni en $q_3$ ni en la cabeza.

### 7.5.2 Paso del encadenamiento de las reglas a las metas.

Supóngase que se tiene un nodo regla correspondiente a una regla $H$ :- $G_1$&... &$G_k$. Se pueden pasar los encadenamientos de la relación suplementaria $S_{i-1}$ al nodo meta hijo correspondiente a $G_i$, de la siguiente manera:

1. primero, se debe determinar qué argumentos de $G_i$ están limitados. Un argumento está limitado si y solo si para todas la variables que aparecen en el están limitadas, es decir que todas las variables de ese argumento son atributos de $S_{i-1}$;

2. sea $G'_i$ el predicado de $G_i$ con todos los argumentos de $G_i$ que están limitados de acuerdo a $S_{i-1}$;

3. convertir las variables de $G'_i$ a argumentos en $S_{i-1}$.

La operación de pasar el encadenamiento de las relaciones suplementarias a las relaciones de encadenamiento, puede ser representada como una regla.

### 7.5.3 Paso del encadenamiento de metas a reglas

Ahora, supóngase que se tiene un nodo meta para la meta $G$, con la relación de encadenamiento $M$. Además supóngase que este nodo meta tiene un nodo regla hijo para la regla $H$ :- $G_1$ & ... & $G_k$ donde la cabeza $H$ se unifica con $G$. El encadenamiento para los argumentos de $G$ también tiene los mismos que los de H. Estos argumentos se encadenan no importando cuales son las variables que aparezcan, y los encadenamientos para las variables se convierten en tuplas para la relación suplementaria 0 , para este nodo regla.

### 7.5.4 Paso de las respuestas de las reglas a las metas.

Cuando un nodo regla produce su relación de respuesta, contribuye a las respuestas de su nodo meta padre. Un nodo regla para una regla con $k$ submetas, procesa la relación suplementaria $S_k$ que encadena todas las variables que son relevantes después de la submeta $k$, es decir , exactamente las variables que aparecen en la cabeza. Sólo se necesita convertir $S_k$ desde el punto de vista de variables al punto de vista de argumentos, aplicado a la cabeza $H$.

Finalmente, se debe notar que la relación resultante para un nodo regla es una de alguna de las relaciones que se tienen como relación resultante de un nodo meta padre. Con orden de producir el resultado para el nodo meta se debe tomar la unión de los resultados para cada nodo regla de ese nodo meta.

## 7.6 Grafo de reglas y metas.

El grafo de reglas y metas, es la representación para una colección de cláusulas de Horn. Se usa para ejecutar la transformación de reglas. En algunos algoritmos de evaluación de búsquedas, se usa para asumir que el predicado $p$ no aparece en una meta durante una expansión descendente, con dos diferentes patrones de encadenamiento. También ayuda a encontrar un buen orden para las submetas en una colección de reglas.

### 7.6.1 Adornos

Cuando, durante la expansión del árbol de metas y reglas, se obtiene un nodo meta:

$p(t_1,\ldots,t_k)$

ese nodo meta está dando una relación de encadenamiento para algún subconjunto de argumentos $p$. Se puede indicar qué argumentos son limitados y cuáles no por un adorno o patrón de encadenamiento, esto es, una cadena de $b$ y $f$ de longitud $k$ si $p$ tiene $k$ argumentos. Si el símbolo $i$ del adorno es $b$ entonces el argumento i del adorno es limitado, esto es, la relación encadenada para el nodo meta incluye un atributo para el argumento $i$. Si el símbolo $i$ del adorno es $f$, entonces, el argumento $i$ de $p$ es libre, esto quiere decir que no hay un argumento para el atributo $i$ en la relación subencadenada para la submeta.

Una submeta que es una instancia del predicado $P$ con una relación de encadenamiento representada por el adorno a se denota como $p^a$.

### 7.6.2 Reglas adornadas

También se necesita indicar el estado de las variables en una regla (limitada o libre). Así como se procesan las submetas de izquierda a derecha durante la expansión de los árboles de reglas y metas, las variables que eran libres vienen a ser limitadas. El algoritmo que indica si una variable se convierte en limitado en la regla:

$r : H :- G_1 \ \& \ \ldots \ \& \ G_k$

es simple:

1. una variable que aparece en un argumento limitado de la cabeza de la regla, es limitado antes de procesar cualquier submeta;

2. una variable es limitada después de procesar la submeta $G_i$ si era limitada antes de procesar $G_i$ o si aparece en alguna parte de $G_i$.

Un adorno para una regla indica que variable es limitada y en que punto y cual es libre. La notación que se usa para el adorno de la regla es de la forma $[X_1,\ldots,X_m|Y_1,\ldots,Y_n]$ donde las $X$ son limitadas y las $Y$ son libres. El adorno para la regla $r$, sobre la consideración de cualquier submeta se muestra como $r_0$, mientras que el adorno que se aplica después de la consideración de la submeta $i$ es $r_i$.

**Ejemplo 7.13**: considérese la regla recursiva ancestro, que se denoto como $r_2$:

$r_2$: anc(X,Y) :- par(X,Z) & anc(Z,Y)

Supóngase que *anc* se llamó con un encadenamiento para el primer argumento, entonces en la regla antes de cualquier consideración se submetas, solamente $X$ es limitada y se representa este hecho por la regla adornada $r_{2.0}^{[X|Y,Z]}$.

La primera submeta, *par(X,Z)* provee un encadenamiento para $Z$, así que la situación después de considerar la primera submeta está representada por $r_{2.1}^{[X,Z|Y]}$. No es necesario representar la situación después de la última submeta, todas las variables serán limitadas, asumiendo que la regla es segura.

### 7.6.3 Construyendo un grafo de reglas y metas.

Se pueden representar los patrones de los encadenamientos que ocurren en el árbol de reglas y metas por una estructura finita llamada, grafo de reglas y metas. Supóngase que se han dado un conjunto de cláusulas de Horn y una meta de búsqueda, si $p$ es el predicado de la búsqueda y a es el adorno que tiene $b$ donde la búsqueda especifica un valor para el correspondiente argumento y tiene $f$ donde no hay un valor especificado, entonces, se inicia la construcción del grafo de reglas y metas para esta búsqueda con el nodo $p^a$. Se considera cada nodo en el grafo de reglas y metas y se expande de acuerdo a las siguientes reglas. Así como se expande se deben agregar nodos metas, que son predicados adornados y los nodos de las reglas, que son nodos representando una regla y algún número de sus submetas para esas reglas. Usando la notación descrita anteriormente se tiene $r_0$, con un adorno para representar la regla $r$ antes de considerar cualquier submeta, y $r_i$ con un adorno para representar la regla $r$ después de considerar sus primeras $i$ submetas .

1. Un nodo meta con un predicado EDB no tiene sucesores.

2. Un nodo meta que es un predicado IDB p con un adorno a tiene sucesores correspondientes a todas las reglas con cabeza de predicado $p$. Si $r$ es una regla entonces $p^a$ tiene el sucesor

   $$r_0^{[X1,\ldots,Xn|Y1,\ldots,Ym]}$$

   donde $X_1,\ldots,X_m$ son todas las variables que aparecen en los argumentos de la cabeza de $r$ que son limitados de acuerdo al adorno a, y $Y_1,\ldots,Y_m$ son las otras variables para $r$.

3. Considérese un nodo regla $r_i^{[X1,\ldots,Xn|Y1,\ldots,Ym]}$, $i \geq 0$ y supóngase que $q(t_1,\ldots,t_k)$ es la submeta $i$+1 de $r$.

   a) Un sucesor de este nodo regla es un nodo meta $q^{\beta}$, ß es el adorno que hace que el argumento $j$ de $q$ sea limitado si todas las variables que aparecen en $t_j$ están entre las variables limitadas de la regla. ß hace al argumento $j$ libre de otra manera.

   b) Si $i$+1 es menor que el número de submetas en la regla $r$, es decir que $q(t_1,\ldots,t_k)$ no es la última submeta de $r$, entonces el nodo

      $$r_i^{[X1,\ldots Xn|Y1,\ldots,Ym]}$$

      tiene un segundo sucesor, el nodo

      $$r_{i+1}^{[X1,\ldots Xm,U1,\ldots,Uj|V1,\ldots V1]}$$

   donde $U_1,\ldots,U_j$ son esas variables entre $Y_1,\ldots,Y_m$ que aparecen en $q(t_1,\ldots,t_k)$ y $V_1,\ldots,V_l$ son las variables restantes de la $Y$.

**Ejemplo 7.14:** considérense las dos reglas siguientes que definen "primos en la misma generación", es decir, individuos en la genealogía que tienen el mismo común ancestro y el mismo número de generaciones de separación. Como un caso especial, cada persona en la base de datos está definida como "su propio primo en la misma generación"; es decir que los hermanos son primos. Se asume que *persona* es una relación unaria EDB conteniendo el conjunto de individuos mencionados en la base de datos y *par(C,P)* es otra relación EDB cuyas tuplas *(c,p)* significan que $p$ es un pariente de $c$.

$r_1$: `sg(X,X) :- persona(X).`

$r_2$: sg(X,Y) :- par(X,Xp) & par(Y,Yp) & sg(Xp,Yp)

la primera regla indica que cada persona está en la misma generación como ella misma y la segunda indica que *X* y *Y* son de la misma generación si tienen padres *Xp* y *Yp*, respectivamente, así que *Xp* y *Yp* son de la misma generación.

Supóngase que la búsqueda es de la forma $sg^{bf}$, que quiere decir, dado un individuo o un conjunto de individuos, encontrar todos los valores *sg(a,b)* de tal manera que "*a*" esté en el conjunto de individuos. La figura 7.10 muestra el grado de reglas y metas que se crea iniciando con esta búsqueda y se sigue con el orden dado de submetas en $r_2$.

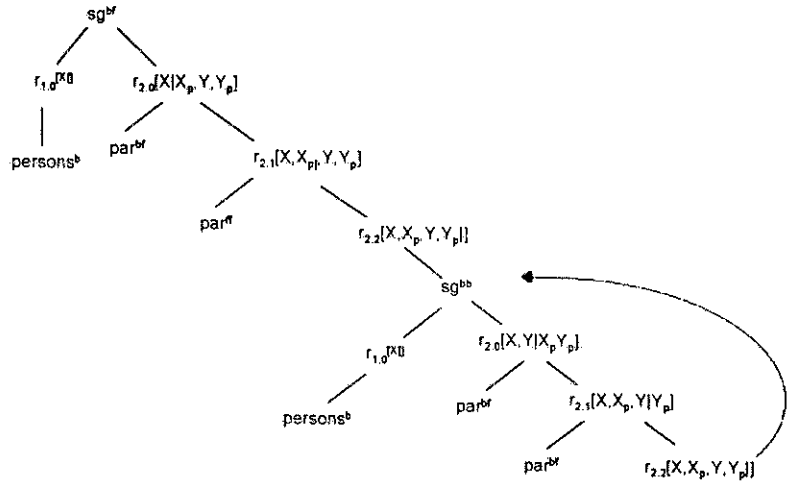



**Figura 7.10 Grafo de reglas y metas para reglas de la misma generación**

Los hijos de la raíz corresponden a las dos reglas para *sg*. En cada caso, el encadenamiento sobre el primer argumento de *sg* provee un encadenamiento para la variable *X* y no otra. Para la $r_1$, el encadenamiento sobre *X* hace que el único argumento de la submeta *persona(X)* sea limitado, como se indicó con el nodo meta $persona^b$, el hijo de $r_{1.0}^{[X|]}$. Nótese que este último nodo pude ser interpretado como " la regla $r_1$ antes de la consideración de cualquier submeta, tiene la variable limitada *X* y no tiene variables libres".

El otro hijo de la raíz

$$r_{2.0}^{[X|Xp,Y,Yp]}$$

tiene el hijo $par^{bf}$, representando la primera submeta de $r_2$. El adorno es *bf*, por el encadenamiento sobre *X* provee un encadenamiento para el primer argumento de *par(X,Xp)*, pero no para el segundo. También tiene el hijo:

$$r_{2.1}^{[X,Xp|Y,Yp]}$$

indicando que después de la primera submeta, *Xp* como *X* son limitados.

El último nodo tiene el hijo $par^{ff}$, el adorno indica que el encadenamiento para $X$ y $Xp$ no provee encadenamiento para ninguno de los argumentos de la segunda submeta, par(Y,Yp). También tiene el hijo

$$r_{2.2}{}^{[X,Xp,Y,Yp|]}$$

que indica que después de procesar las dos primeras submetas de $r_2$, se tiene un encadenamiento para todas las variables. Así este nodo tiene el hijo $sg^{bb}$ que representa la tercera y última submeta de $r_2$ y sus dos argumentos son limitados.

Ahora se tiene un segundo nodo para el predicado $sg$, ahora con el adorno $bb$, se expande de la misma manera que el primer nodo, $sg_{bf}$. El adorno del nodo regla es diferente pero se obtiene la tercera submeta, $sg^{bb}$ nuevamente. Así, no se necesitan agregar más nodos y la construcción del grafo de reglas y metas está completa.

El orden que se siguió para las submetas en $r_2$ no fue precisamente el mejor que se pudo escojer. La razón es que cuando se inició con $sg^{bf}$ se forzó a evaluar $par^{ff}$ y se incluyó la relación $par$ completamente para evaluar $r_2$. Intuitivamente se podría evitar la evaluación de submetas que no tienen por lo menos un argumento limitado ya que típicamente el encadenamiento de un argumento rebajará significativamente el número de tuplas. Este suceso para el mismo ejemplo de la misma generación, se puede evitar examinar completamente la relación $par$ si se mueve la submeta $sg$ al medio del cuerpo; es decir que se reescriben las reglas como:

$r_1$: $sg(X,X)$ :- $persona(X)$.
$r_2$: $sg(X,Y)$ :- $par(X,Xp)$ & $sg(Xp,Yp)$ & $par(Y,Yp)$.

El grado de reglas y metas para estas reglas y la búsqueda $sg^{bf}$ se muestra en la figura 7.11.

## 7.7 Un único patrón de encadenamiento

De un grafo de reglas y metas, se puede determinar un patrón de encadenamiento con el que cada predicado es llamado. Con esta información se puede dividir los predicados, cada predicado IDB tiene un único patrón desencadenamiento asociado. La técnica es simple.

1. Para cada adorno a de un predicado IDB $p$, tal que $p^a$ aparece en el grafo de reglas y metas, se crea un nuevo predicado $p_a$.

2. Para cada una de las reglas $p$, dígase $r$, hacer una copia $r_a$ de la regla con $p_a$ como la cabeza del predicado.

3. Examinar el nodo regla hijo $r_0$ del nodo meta $p^a$. Extendiéndose a la derecha de un nodo $r_0$ se tiene una cadena de nodos llamada $r_1,\ldots,r_k$ donde $k$ es el número de submetas de $r$. Esto tendrá una colección de nodos meta hijos correspondientes a las submetas de $r$.

   a) Si $q^\beta$ es un nodo meta y $q$ es un EDB o un predicado construido, dejar la submeta correspondiente en $r_a$ como se encuentra en $r$.

   b) Si $q$ es un predicado IDB, entonces cambiar el predicado en la submeta correspondiente de $r_\beta$ a $q_\beta$.

**Ejemplo 7.15:** considérense las reglas de la misma generación del ejemplo 7.14 y el grafo de reglas y metas de la figura 7.10. El predicado IDB *sg* tiene dos nodos metas con los adornos *bf* y *bb*. Se deben crear dos predicados, *sg_bf* y *sg_bb*. La primera regla $r_1$, tiene solamente la submeta EDB *persona*, así que las dos copias de $r_1$ son la misma, excepto por el predicado de la cabeza.

La regla $r_2$ tiene una submeta EDB, *sg(Xp,Yp)* y donde el predicado cabeza es *sg_bf* o *sg_bb*, el adorno para esta submeta es limitado, en la figura 7.10, para ser *bb*. Así los cuerpos de las dos versiones de $r_2$, que se llama $r_2$*_bf* y $r_2$*_bb*, ambas tienen la submeta *sg_bb(Xp,Yp)*. Entonces el nuevo conjunto de predicados y reglas se muestra en la figura 7.12.

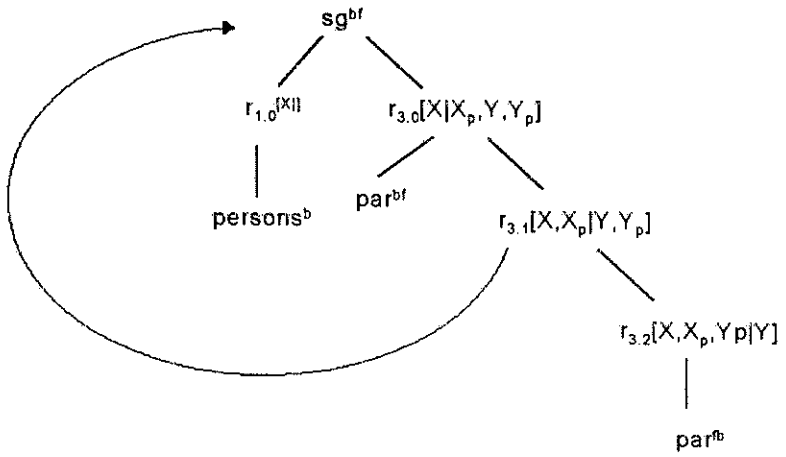



**Figura 7.11 Grafo mejorado de reglas y metas después de reordenar las submetas.**

Ocurre que se debe construir el árbol de reglas y metas con la meta raíz *sg_bf* y un encadenamiento para el primer argumento, entonces, los adornos indican el patrón de encadenamiento de los nodos meta. Esto es, todas las ocurrencias de un nodo meta con predicado *sg_bf* tienen el patrón de encadenamiento *bf*, y todas las ocurrencias de *sg_bb* tienen ambos argumentos limitados. En general todos los nodos con el predicado *p_a* en un árbol de reglas y metas tendrán, por lo menos, los encadenamientos indicados por a, pero en algunas situaciones , pueden existir otros encadenamientos.

```
r1_bf: sg_bf(X,X) :- persona(X).
r1_bf: sg_bf(X,Y) :- par(X,Xp) & par(Y,Yp) & sg_bb(Xp,Yp).

r2_bb: sg_bb(X,X) :- persona(X).
r2_bb: sg_bb(X,Y) :- par(X,Xp) & par(Y,Yp) & sg_bb(Xp,Yp).
```

**Figura 7.12 División de reglas para la misma generación.**

### 7.8 Reorden de las submetas

Cuando se construye el grafo de reglas y metas , se tiene la opción de reordenar las submetas de una regla así como se expanden los nodos de la regla correspondientes. Si se crea un nuevo predicado *p_a* para cada adorno *a* de cada predicado original *p*, entonces, se pueden reordenar las submetas de las reglas para *p_a* como se desea, independientemente del orden de las submetas correspondientes en cualquier variante de *p*. En efecto, aunque no se divida *p*, se puede procesar el grafo de reglas y metas de tal manera que el diferente orden de las submetas se usa para distintos ejemplos de la misma regla. La justificación es que el punto fijo mínimo semántico de las reglas no depende del orden en que se listen las submetas.

El problema de ordenar las submetas en una regla es similar al problema de ordenar la unión de varias relaciones. Sin embargo, cuando se evalúan reglas no se tienen relaciones disponibles para todas las submetas inicialmente. A través del proceso de pasar información se desarrollaron encadenamientos para algunas de las submetas y solamente entonces se trato de generar la porción relevante de la relación para esa submeta. Para hacerlo de otra manera, se debe procesar la relación entera para algún predicado IDB, cuando lo que se requiere es una pequeña porción de la relación.

Además, es posible que alguna submeta pueda ser evaluada solamente con algunos patrones de encadenamiento. Algunos ejemplos de estos fenómenos son los siguientes:

1. los predicados construidos usualmente requieren algún encadenamiento. Por ejemplo, la submeta $X = Y$ no puede ser evaluada a menos de que $X$ y $Y$ sean limitadas. Si se dice que $X$ es limitada para el conjunto de valores $\{a_1,\ldots,a_n\}$, entonces se puede usar la relación finita $\{(a_1,a_1),\ldots,(a_n,a_n)\}$, para la submeta $X = Y$. Si $X$ y $Y$ no son limitadas, entonces existe un conjunto infinito de pares que hacen la submeta verdadera;

2. similarmente, la submeta $X < Y$ no puede ser evaluada a menos de que $X$ y $y$ sean limitadas; de otra forma un conjunto infinito de pares satisface esta submeta;

3. se quiere declarar que las relaciones EDB no pueden ser evaluadas a menos de que exista un atributo limitado sobre el que la relación EDB tenga un índice. Si no existe tal atributo, entonces se debe examinar la relación completa aunque existan algunos argumentos limitados cuyos atributos correspondientes no tengan índices;

4. un predicado IDB cuyas reglas involucran funciones de símbolos puede ser evaluada si ciertos argumentos son limitados, pero no si son libres.

**Ejemplo 7.16:** supóngase que se tiene una base de datos de buenos elementos, es decir una relación unaria *bueno*. La construcción de listas se hace usando la función de símbolos binaria *cons*. Es decir que, *cons(H,T)* representa la lista con elementos *H* de cabeza y los elementos restantes *T*. Se usará la constante *nulo* para representar la lista vacía. Entonces se pueden definir listas buenas con las reglas

$r_1$: buenaLista(nulo).
$r_2$: buenaLista(cons(H,T)) :- bueno(H) & buenaLista(T).

esto es, la lista vacía es buena, y una lista es buena si el primer elemento *H* es *bueno* y el balance de la lista, representado por T en $r_2$ es una lista buena.

Por ejemplo, las listas *a*, *b* y *c* pueden ser representadas por el término

*cons(a,cons(b,cons(c,nulo)))* (IV)

asúmase que a, *b* y *c* están en la relación para *bueno*. Entonces, la regla $r_2$ indica que a, *b* y *c* son una lista buena si *b* y *c* son una lista buena. Además, *b*, *c* son buenos si la lista *c* es buena. Finalmente, $r_2$ dice que *c* es una buena lista si la lista vacía es buena, y $r_1$ afirma que la lista vacía es buena.

Si se hace una búsqueda de la forma $buenaLista^b$, una búsqueda con los argumentos (IV) entonces, se puede responder la búsqueda. Por ejemplo, se puede deducir por la expansión del árbol de reglas y metas que (IV) es una buena lista, a, *b* y *c* son buenos. Sin embargo, si se quiere la búsqueda buenaLista(X), es decir todas las listas buenas, se tiene un ejemplo de la meta adornada $buenaLista^f$. El resultado de esta búsqueda es infinito cuando la relación EDB *bueno* no se encuentra vacía. Así, no se puede tener el árbol de expansión o cualquier otro algoritmo de evaluación de búsquedas, para regresar una respuesta.

La conclusión es que mientras se pueda permitir una submeta con un patrón de encadenamiento $buenaLista^b$, no se puede permitir el patrón $buenaLista^f$.

## 8. CONJUNTOS MÁGICOS

Los conjuntos mágicos son una técnica de reescritura de las reglas de la lógica, de tal manera que la evaluación ascendente ofrece todas las ventajas asociadas con la evaluación descendente. Con esta aproximación, las reglas y la búsqueda son usadas para construir nuevas reglas que respondan la búsqueda, aunque generalmente no son iguales a las reglas originales.

### 8.1 Técnica de reescritura de reglas.

Considérense las siguientes reglas para definir ancestros en términos de una relación EDB *par*, donde *p(C,P)* significa que *P* es pariente de *C*.

$r_1$ : *anc(X,Y) :- par(X,Y)*
$r_2$ : *anc(X,Y) :- par(X,Y) & anc(Z,Y)*

en palabras, $r_2$ dice que *Y* es ancestro de *X* si no existe una *Z* tal que *Z* es padre de *X* y *Y* y es un ancestro de *Z*. Supóngase la relación *P* para el predicado *par* se muestra en la figura 7.8, las aristas representan los pares *(x,y)* en *P*, con el hijo abajo y el padre arriba. Si se inicia con la meta *anc(j,W)*, es decir, encontrar todos los *j* ancestros, esta meta se puede mostrar como la raíz de un árbol de metas y reglas. Se puede explorar, entonces, el árbol de reglas y metas para hacer una búsqueda descendente eficientemente, considerando únicamente los individuos que son ancestros de *j*. En comparación de los algoritmos de evaluación ascendente que deben construir la relación completa para *anc*. Si la mayoría de los individuos mencionados en la relación *par* no son ancestros de *j*, la aproximación ascendente tomará mucho más tiempo que la aproximación descendente, sin embargo existe un número de razones por las que la aproximación ascendente es preferible a la descendente.

1. Si no se tiene cuidado con la forma de como se hacen los cálculos de la aproximación descendente se puede caer en ciclos infinitos y nunca obtener la respuesta, esto puede suceder cuando la relación contiene ciclos.

2. Aunque se tenga cuidado con los cálculos, saber cuando terminar con los ciclos, en las reglas de Datalog, no es fácil.

3. Aunque se trabaje con el árbol de metas y reglas, la expansión es posible para la misma submeta en el árbol provocando así significantes cantidades de trabajo repetido.

4. Los algoritmos ascendentes pueden hacer uso eficiente de las técnicas para las uniones de relaciones masivas. (Hay que recordar que la unión es el paso central para la operación EVAL, que en efecto es el corazón de la evaluación semi ascendente). En comparación, la evaluación descendente tiende a ser adecuada con relaciones pequeñas, cada una asociada con los nodos del árbol de reglas y metas.

5. Los algoritmos descendentes requieren de unificación pero los algoritmos ascendentes sólo necesitan la igualación de términos, que es una operación más simple.

### 8.2 La única propiedad de encadenamiento.

El algoritmo de modificación de reglas está cercanamente relacionado con la forma en que se organiza la evaluación de los árboles de reglas y metas. La idea principal es que no se ha considerado cuando un dato *p(m)* es verdadero, para el

predicado IDB *p* y para la tupla m, tampoco se ha establecido que tal valor pueda ser de interés, en el sentido de que m es una tupla de la relación de encadenamiento para uno de los nodos de las metas de *p*.

En orden para la notación del patrón de encadenamiento de un predicado, se requiere que las reglas tengan la única propiedad de encadenamiento con respecto a las metas dadas. Un conjunto de reglas , tiene la única propiedad de encadenamiento, si cuando se construye el grafo de reglas y metas iniciando con la meta adornada respecto al orden en que han sido escritas las submetas, no predicados IDB aparecen con dos adornos diferentes.

**8.3 Predicados Mágicos.**

Asumiendo la única propiedad de encadenamiento, se crea para cada predicado IDB *p* un predicado mágico, que se llamará *m_p*. Los argumentos de los predicados mágicos de *p* corresponden a los argumentos que hacen que del único patrón de encadenamiento para *p* sea limitado. Por ejemplo, si el patrón de encadenamiento par *p* es *bfbf*, entonces, *m_p* es una relación binaria cuyas tuplas están encadenadas para el primer y tercer argumento de *p*.

El sentido para *m_p*, es que *m_p(v)* debe ser verdadero si y sólo si, en el árbol de reglas y metas, *v* es un encadenamiento pasado a un nodo meta con predicado *p*. Esto no es exactamente correcto, porque en el árbol de reglas y metas deben encontrarse conjuntos que cubran más argumentos que el supuesto único patrón de encadenamiento para *p*. Cualquier encadenamiento adicional es ignorado en los predicados mágicos, pero hace que la evaluación de las reglas resultantes sea menos eficiente de lo que puede ser.

**8.4 Predicados Suplementarios.**

También se usan predicados que representan las relaciones suplementarias, que se usan para pasar la información, en un grafo de reglas y metas. Si *r* es una regla con *k* submetas, se deben usar los predicados $sup_{r.i}$ para $i = 0,1,\ldots,k-1$. Los argumentos de $sup_{r.i}$ son variables de la regla *r*, y corresponden a los argumentos de la relación suplememtaria *i*, para la regla *r*. Es decir, que son todas las variables de *r* que son:

1. limitadas en la cabeza, de acuerdo al único patrón de encadenamiento para la cabeza, o;

2. aparecen en una de las primeras *i* submetas de *r* y también aparecen en una submeta posterior o en la cabeza.

Nótese que no se usa la última relación suplementaria, prefiriendo ir directamente desde la última submeta a la cabeza.

Como usualmente se refiere a las reglas como $r_1,\ldots,r_n$, se debe adoptar la siguiente convención. Si una enumeración de las reglas se sobre entiende, entonces *supj.i* se encontrará para el predicado suplementario *i*, para la regla $r_j$; es decir que $sup_{rj}$ y $sup_{j.i}$ serán tratados como sinónimos, esta ambigüedad no debe causar dificultas al momento de la práctica.

**8.5 Transformación básica de los conjuntos mágicos.**

Se da ahora una forma mecánica para convertir un conjunto de reglas dadas y una búsqueda en un nuevo conjunto de reglas, con un predicado cuya relación es la respuesta de la búsqueda. Este conjunto de reglas resultante debe minimizar el árbol de expansión de reglas y metas. Así, las nuevas reglas deben ser correctas ya que producen la misma respuesta que el árbol de expansión y relativamente eficiente.

**Algoritmo 8.1** Transformación de un conjunto mágico de reglas.

**Entrada:** una colección de reglas y una búsqueda $q(s_1,\ldots,S_n)$. Se asume que las reglas tienen la propiedad de único encadenamiento con respecto a la búsqueda.

**Salida:** un nuevo conjunto de reglas de tal manera que la relación *q* es la respuesta para la búsqueda.

**Método:** Se inicia creando los siguientes nuevos predicados.

i) Predicados mágicos para los predicados IDB. Para cada predicado IDB *p*, *m_p* es el predicado mágico para *p*. Los argumentos de *m_p* son los argumentos limitados de *p*; hay que recordar que el patrón de encadenamiento para cada predicado IDB se asume único.

ii) Predicados suplementarios. Para cada regla $r_j$, con *k* submetas, se crean predicados suplementarios $sup_{j.i}$, para $i= 1,\ldots k-1$.

Ahora se crean las reglas para cada uno de estos nuevos predicados, y se crean nuevas reglas para los predicados originales IDB.

1. Reglas para los predicados mágicos. Sea *p* un predicado IDB, y considérese cada una de las submetas de las reglas dadas que tienen el predicado *p*. Supóngase que *p* aparece en la submeta *i* de la regla $r_j$, y que esta submeta es $p(t_1,\ldots,t_n)$. Sean los argumentos para la relación suplementaria *i-1* las variables $U_1,\ldots,U_1$. Finalmente, sea $i_1,\ldots,i_n$, la lista de argumentos de *p* que están limitados en el patrón de encadenamiento. Entonces se crea la regla

$$m_p(t_{i1},\ldots,t_{im}) \text{ :- } sup_{j.i-1}(U_1,\ldots,U_1)$$

Por ejemplo, supóngase que la submeta en cuestión es *p(f(X),Y)* y los argumentos del predicado suplementario *i-1* para la regla $r_j$, son *X*, *U* y *V*. Entonces, el primer argumento de *p* está limitado y el segundo está libre. Así, *m_p* tiene sólo un argumento, y una de las reglas para *m_p* puede ser

$$m_p(f(X)) \text{ :- } sup_{j.i-1}(X,U,V)$$

2. Reglas para los predicados suplementarios cero. Sea $p(t_1,\ldots,t_n)$ la cabeza da la regla $r_j$, y, supóngase que $i_1,\ldots,i_m$ son argumentos limitados de *p*. Además, Sea $X_1,\ldots,X_1$ las variables que están limitadas por la cabeza de $r_j$, Es decir, las variables que aparecen entre $t_{i1},\ldots,t_{im}$. entonces la regla que define $sup_{j.0}$ es

$$sup_{j.0}(X_1,\ldots,X_1) \text{ :- } m_p(t_{i1},\ldots,t_{im}).$$



PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

Por ejemplo, sea $p(f(X,Y),Z)$ la cabeza de $r_j$, y, supóngase que sólo el primer argumento de $p$ está limitado. Entonces, la relación suplementaria cero para $r_j$ tiene argumentos correspondientes a las variables $X$ y $Y$, y la regla es

$$sup_{j.0}(X,Y) \text{ :- } m_p(f(X,Y)).$$

3. Reglas para otros predicados suplementarios. Para cada regla $r_j$, con $k$ submetas y para cada $i = 1,2...,k-1$, se crea una regla simple para $sup_{j.i}$. Sean las variables de $r_j$, $U_1,...,U_l$ que son atributos de la relación suplementaria $i-1$, $r_j$ y sean $V_1,...,V_m$, las variables que son atributos de la relación suplementaria $i$ para $r_j$. Finalmente, sea $p(t_1,...,t_n)$ la submeta $i$ de $r_j$. Entonces se tiene la regla

$$sup_{j.i}(V_1,...,V_m) \text{ :- } sup_{j.i-1}(U_1,...,U_t) \ \& \ p(t_1,...,t_n).$$

Nótese que las $V$ siempre aparecen entre las $U$ o entre los argumentos de $p$, así que la regla anterior es segura. por ejemplo, Supóngase que la submeta $i$ es $p(f(X))$, las variables limitadas por $sup_{j,i-1}$ son $W$, $X$ y $Z$, mientras que las limitadas por $sup_{j.i}$ son $W$, $Y$ y $Z$. Entonces la regla para $sup_{j.i}$ es

$$sup_{j.i}(W,Y,Z) \text{ :- } sup_{j.i-1}(W,X,Z) \ \& \ p(f(X),Y).$$

4. Reglas para los predicados IDB. Sea $p$ un predicado IDB y $r_j$ una de las reglas para $p$. Finalmente, supóngase que $r_j$ tiene $k \geq 1$ submetas. Entonces, se obtienen tuplas para el predicado suplementario $(k-1)$ y la última submeta. Esto es, si $U_1,...,U_l$ son variables que son atributos de la relación suplemetaria $(k-1)$ para $r_j$, $h(t_1,...,t_n)$ es la cabeza de $r_j$ y la submeta $k$ de $r_j$ es $p(v_1,...,v_m)$, entonces, se tiene la regla

$$h(t_1,...,t_n) \text{ :- } sup_{j.k-1}(U_1,...,U_l) \ \& \ p(v_1,...,v_m).$$

Por ejemplo, supóngase a $r_j$ como:

$$h(f(X),Y) \text{ :- } ... \ \& \ p(Z,g(W,Y)).$$

donde existen $k$ submetas en total, y las variables que aparecen en la relación suplementaria $k-1$ son W, $X$ y $Y$. Entonces una de las reglas para el predicado IDB $h$ es

$$h(f(X),Y) \text{ :- } sup_{j.k-1} \ \& \ p(Z,g(W,Y)).$$

Existe una excepción para la regla (4) cuando se tienen cero submetas en una regla, en tal caso, no existe la relación suplementaria $k-1$, y se construye la cabeza de $sup_{j.0}$ solamente; es decir que la regla es

$$h(t_1,...,t_n) \text{ :- } sup_{j.0}(U_1,...,U_l).$$

donde $U_1,...,U_l$ son los argumentos del predicado suplementario cero para la regla $r_j$.

5. La inicialización de reglas. Sea la búsqueda $q(s_1,...,S_n)$ y supóngase que $i_1,...,i_m$ son los argumentos limitados de la

búsqueda; es decir que $s_{i1},\ldots,s_{im}$ son términos fundamentales. Entonces, se tiene la regla

$m_p(s_{i1},\ldots,s_{im})$.

Intuitivamente, esta regla indica que se debe tener interés en las tuplas de $q$ que encajan con las constantes de la búsqueda.

Hay que notar que únicamente las reglas del V grupo dependen de la búsqueda. Las otras reglas dependen del patrón de encadenamiento de la búsqueda aunque no de las constantes particulares involucradas. A las reglas generadas por los cinco grupos anteriores se les puede referenciar como conjunto de reglas mágicas o reglas mágicas.

**Ejemplo 8.1:** considérense las reglas originales para " la misma generación"

$r_1$ sg(X,X) :- persona(X).
$r_2$ sg(X,Y) :- par(X,Xp) & sg(Xp,Yp) & par(Y,Yp).

con la búsqueda *sg(a,W)*; es decir, que trata de encontrar los primos del individuo *a*. Nótese que se han reordenado las submetas para obtener el grafo de reglas y metas de la figura 7.11.

Las reglas de los cinco grupos se muestran en la figura 8.1. Por ejemplo, la regla (1) aparece porque la segunda submeta de $r_2$ es una submeta *sg*. En ese punto, en la regla, sólo *X* y *Xp* están limitadas y solamente el primer argumento de *sg* está limitado. En efecto *bf* es el único adorno asociado con *sg* en esa regla, así que se sabe que el predicado mágico para *sg* tendrá solamente un argumento.

GRUPO I

(1) $m_sg(Xp) :- sup_{2.1}(X,Xp)$.

GRUPO II

(2) $sup_{1.0}(X) :- m_sg(X)$.
(3) $sup_{2.0}(X) :- m_sg(X)$.

GRUPO III

(4) $sup_{2.1}(X,Xp) :- sup_{2.0}(X)$ & par(X,Xp).
(5) $sup_{2.2}(X,Xp) :- sup_{2.1}(X,Xp)$ & sg(Xp,Yp).

GRUPO IV

(6) sg(X,X) :- $sup_{1.0}(X)$ & persona(X).
(7) sg(X,Y) :- $sup_{2.2}(X,Yp)$ & par(Y,Yp).

GRUPO V

(8) m_sg(a).

**Figura 8.1 Conjunto Mágico, transformación de reglas**

En la regla (1), el argumento de *m_sg* es una copia del primer argumento de la submeta *sg* en $r_2$, y los argumentos de $sup_{2.1}$ son variables limitadas después de la primera submeta de $r_2$.

Para cada una de las tuplas agregadas a la relación IDB de la figura 8.1, se puede mostrar, por inducción, el número de ciclos de la evaluación ascendente, que:

1. las relaciones *m_sg*, . $sup_{1.0}$ y $sup_{2.0}$ cada una representa el conjunto de los ancestros del individuo *a;*

2. la relación suplementaria $sup_{2.1}$, es el conjunto de la relación EDB *par* tal que el primer argumento es un ancestro de *a;*

3. la relación suplementaria $sup_{2.2}$ es el conjunto de pares *(X,Yp)*, tal que *X* es un ancestro de *a* y *Yp* es un primo de *X*, pero *Yp* está en una generación anterior a la de *X;*

4. las únicas tuplas agregadas a *sg* son aquellos pares *(X,Y)* tal que son primos de la misma generación, y *X* es un ancestro de *a;*

Así las reglas modificadas evitan inferir cualquier valor de *sg* que puedan derivar los valores de la forma *sg(a,b)* para alguna *b*. En comparación, las reglas originales derivan valores *sg(c,d)*, donde *c* no es un ancestro de *a*. Tal valor no se usa para probar cualquier valor de la forma *sg(a,b)* y es no es necesario para la respuesta de la búsqueda *sg(a,W)*.

Nótese que las reglas de la figura 8.1 pueden ser simplificadas. Por ejemplo se puede sustituir *m_sg* por $sup_{1.0}$ y $sup_{2.0}$.

**Ejemplo 8.2:** Ahora considérense las funciones de símbolos basadas en las reglas para un buena lista

$r_1$ : gl(nulo).
$r_2$ : gl(cons(H,T)) :- g(H) & gl(T).

aquí se usa *g* para *bueno* y *gl* para *buenaLista*, para hacer las reglas más concisas. La búsqueda es $gl(l_0)$ donde $l_0$ es una lista dada, no importa cual. Un grafo de reglas y metas apropiado para estas reglas, se muestra en la figura 8.2.

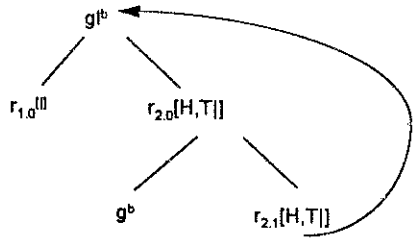



**Figura 8.2 Grafo de reglas y metas para las reglas de una buena lista.**

Las reglas generadas por el algoritmo 8.1 se muestran en la figura 8.3. Algo que hasta el momento no se ha contemplado se observa en la regla (2) donde el predicado $sup_{1.0}$ no tiene argumentos. En efecto $sup_{1.0}$ denota verdadero o falso,

y la regla (2) indica que su valor es verdadero si *nulo* se encuentra en la relación *m_gl*, y *falso* si es de otra manera. También hay que notar que la regla (5) es un ejemplo para la excepción del grupo IV, un regla con cero submetas, donde la cabeza se obtiene de la relación suplementaria cero.

Se puede ver lo siguiente, por inducción sobre el número de ciclos de la evaluación ascendente aplicado a las reglas de la figura 8.3

1. La relación *m_gl* consiste en los sufijos de $l_0$ que están precedidos solamente por buenos elementos; en particular está en *m_gl* si y sólo si todos los elementos de $l_0$ son buenos.

2. La relación $sup_{2.0}$ tiene la tupla *(H,T)* si y sólo y si la lista *con(H,T)* es un sufijo de $l_0$ y está precedido por buenos elementos, mientras que $sup_{2.1}$ es el subconjunto de $sup_{2.0}$ definido por la condición adicional de que *H* también sea buena.

3. La relación *gl* está vacía si $l_0$ no consiste, exclusivamente, en buenos elementos. Si todos los elementos de $l_0$ son buenos, entonces, *gl* consiste de todos los sufijos de $l_0$.

GRUPO I

(1) m_gl(T) :- $sup_{2.1}$(H,T).

GRUPO II

(2) $sup_{1.0}$() :- m_gl(nulo).
(3) $sup_{2.0}$(H,T) :- m_gl(cons(H,T)).

GRUPO III

(4) $sup_{2.1}$(H,T) :- $sup_{2.0}$(H,T) & g(H).

GRUPO IV

(5) gl(nulo) :- $sup_{1.0}$().
(6) gl(cons(H,T)) :- $sup_{2.1}$(H,T) & gl(T).

GRUPO V

(7) m_gl($l_0$).

**Figura 8.3 Conjunto mágico, transformación de reglas para una buena lista.**

**Ejemplo 8.3:** Considérense las reglas

$r_1$ : p(X,Y,D) :- a(X,Y,D).
$r_2$ : p(X,Y,D) :- p(X,Z,E) & p(Z,Y,F) & D = E+ F.

Estas reglas tienen dos submetas recursivas en un cuerpo y también una submeta aritmética.

*a* es un predicado EDB y *a(X,Y,D)* significa que existe un arco del nodo *X* al nodo *Y* de longitud *D*. El predicado IDB *p(X,Y,D)*, similarmente, significa que existe un camino de longitud *D* de X a *Y*.

La tercera submeta de $r_2$, $D = E + F$, no es un tipo que se haya mencionado con anterioridad. Sin embargo se debe pensar como si fuera una submeta *s(D,E,F)*, con los patrones de encadenamiento *fbb* y *bbb*. Las relación EDB *s* consiste del conjunto infinito de tres valores, de tal manera que el primer elemento es la suma del segundo y el tercero. Como con un predicado construido, se puede unir la relación para *s* con relaciones para otras submetas después de haber encontrado un conjunto finito de valores para el segundo y tercer argumento, es decir después de que *E* y *F* sean limitados. Así las reglas se pueden escribir

*r1: p(X,Y,D) :- a(X,Y,D).*
*r2: p(X,Y,D) :- p(X,Z,E) & p(Z,Y,F) & s(D,E,F).*

Supóngase que la búsqueda es $p^{bff}$, es decir encontrar todos los nodos alcanzables de un nodo dado $x_0$, y, encontrar la longitud de cada camino a cada nodo. Un grafo de reglas y metas para estas reglas y búsqueda se muestra en la figura 8.4. No es el único grafo de reglas y metas, ya que se pudo hacer un reordenamiento de las submetas en $r_2$. Sin embargo, la submeta *s* solamente puede ser la tercera ya que sólo entonces *E* y *F* pueden ser encontradas.

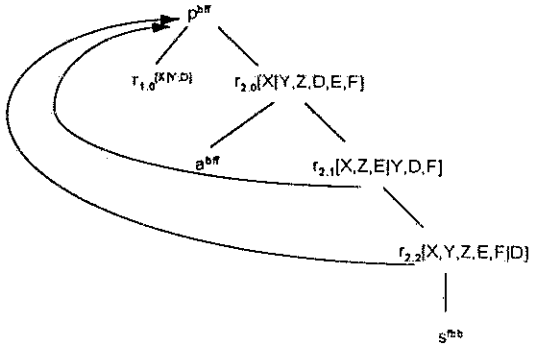



**Figura 8.4 Grafo de reglas y metas para las reglas de caminos y longitudes.**

Hay que estar alertas, ya que aunque las reglas anteriores son de Datalog, es decir que no hay funciones de símbolos, se espera que la respuesta sea infinita si los arcos tienen ciclos. La razón es que *s*, aunque se trate como una relación EDB es en realidad infinita. Cuando existen ciclos, la evaluación ascendente de las reglas pueden ser aplicada a $r_2$ un número infinito de veces. En cada ciclo, la relación para *p* es finito, ya que se generan nuevos valores para la longitud en cada ciclo. y las reglas nunca convergen. En realidad la operación de suma es como una función de símbolos así que las reglas no son de Datalog. Sin embargo, se puede aplicar la transformación de conjuntos mágicos a las reglas anteriores y obtener un programa de lógica como el que se muestra en la figura 8.5. Si "*a*" es una relación finita representando un grafo acíclico, entonces, estas reglas convergen para la evaluación ascendente así como el conjunto de reglas $\{r_1, r_2\}$, pero, si *a* tiene ciclos, se espera que las reglas nunca converjan.

```
        GRUPO I

(1) m_p(X) :- sup2.0(X).
(2) m_p(Z) :- sup2.1(X,Z,E).

        GRUPO II

(3) sup1.0(X) :- m_p(X).
(4) sup2.0(X) :- m_p(x).

        GRUPO III

(5) sup2.1(X,Z,E) :- sup2.0(X) & p(X,Z,E).
(6) sup2.2(X,Y,E,F) :- sup2.1(X,Z,E) & p(Z,Y,F).

        GRUPO IV

(7) p(X,Y,D) :- sup1.0(X) & a(X,Y,D).
(8) p(X,Y,D) :- sup2.2(X,Y,E,F) & s(D,E,F).

        GRUPO V

(9) m_p(x0).
```

**Figura 8.5 Conjuntos mágicos, transformación para las reglas de caminos.**

## 8.6 Recursividad lineal.

Hay algunos casos donde las reglas mágicas, tienen una recursividad lineal, es decir que existe por lo menos una submeta de alguna regla que es mutuamente recursiva con la cabeza.

La recursividad lineal derecha, tiene la submeta recursiva localizada en la parte derecha del cuerpo de la regla, es decir la última.

La recursividad lineal izquierda, tiene la submeta recursiva localizada en la primera de la lista de submetas.

## 9. SISTEMAS DE BASES DE DATOS BASADOS EN LA LÓGICA.

Hay que recordar que los sistemas de bases de datos basados en la lógica tienen un sistema de programación que:

1. tiene un lenguaje declarativo, esto es, la lógica en una se sus posibles formas, sirviendo como un lenguaje anfitrión y un lenguajes de búsquedas;

2. soporta las principales capacidades de un sistema de base de datos, es decir, eficiente acceso a las cantidades masivas de datos, compartir datos, acceso concurrente a los datos y resistente a fallas.

A principios de los años 90, no existía un sistema comercial de esta naturaleza, pero, existe actualmente un número de experimentos en progreso, en esta dirección. La idea aquí es mostrar un ejemplo de estos proyectos. Hay que tomar en cuenta que las descripciones no deben encajar exactamente con las definidas en este trabajo.

### 9.1 Aplicaciones de sistemas basados en la lógica.

En general las áreas de aplicación, para este tipo de sistemas, son sistemas con cantidades masivas de datos y que necesitan una facilidad más poderosa que los típicos lenguajes de bases de datos, tales como SQL. Hay que recordar que tales lenguajes de bases de datos tienen, esencialmente, el poder de usar reglas de la lógica no recursivas con negación permitida en sus submetas. Pero, no tienen el poder de la aplicación recursiva de las reglas. Así, las aplicaciones requieren un sistema de bases de datos basado en la lógica si se tiene una estructura recursiva o anidada que se necesita consultar. Las clases generales de este tipo de aplicaciones incluyen:

1. diseño de base de datos donde los objetos se componen jerárquicamente. Un importante ejemplo es una base de datos VLSI CAD (Computer Aided Design) donde los chips son diseñados como celdas , las celdas están compuestas de subceldas y, así, sucesivamente, a un nivel finito que depende de un diseñó particular. La expansión de un diseño donde se pueda ver o procesar la representación completa de un objeto, es una operación definida recursivamente y no se puede hacer en un lenguaje de búsquedas típico de un sistema de base de datos;

2. una base de datos CASE (Computer Aided Software Engineering), donde los programas son representados, típicamente, como estructuras recursivas tales como un árbol de análisis. Otro aspecto de la representación de programas es la graficación y requiere la búsqueda de gráficas de tamaño arbitrario, otra operación que no puede hacerse en un lenguaje de búsquedas típico. Un ejemplo de tales gráficas, es el diagrama de flujo usado en un programa de análisis de flujo de datos. Otro ejemplo es un diagrama de dependencias, indicando como cambios de una pieza de código, afectas a otras piezas de código, documentación, etc;

3. Patrones de reconocimiento de bases de datos, donde los datos iniciales, tal vez bits, se agrupan en estructuras progresivas más complejas. Un ejemplo es una base de datos para representar fotografías, de terrenos por ejemplo. Se quiere agrupar bits en líneas y curvas y luego agrupar líneas y curvas en simples patrones, colecciones de patrones similares en regiones y así, sucesivamente. Un problema de patrón de reconocimiento similar involucra cadenas de proteínas que mejoran un gen. En muchos casos se puede usar la descripción de estructuras

gramaticales de acuerdo a una definición recursiva, y , nuevamente, las búsquedas recursivas que describen patrones no se pueden manejar con un lenguaje de búsquedas de base de datos típico.

**Ejemplo 9.1:** uno de los ejemplos discute una base de datos de trabajos de arte, que puede ser una base de datos de CAD. Aquí se tiene una relación EDB *conjunto(I,X,Y)* que significa que la celda *I* tiene el punto *(X,Y)*. Otra relación EDB *contiene(I,J,X,Y)* indica que la cela *I* contiene una copia de la celda *J* con un origen en el punto *(X,Y)* en el sistema de coordenadas de *I*. Entonces, se puede definir una relación IDB *en(I,X,Y)* que significa que en la celda *I*, considerándose todas las subcelas y sus subceldas, etc., existe el punto (X,Y). Las reglas que definen a *en* son

*en(I,X,Y) :- conjunto(I,X,Y).*
*en(I,X,Y) :- contiene(I,J,U,V) & en(J,W,Z) & X=U+W & Y=V+Z.*

Las dos últimas submetas no son de la forma permitida ya que no solamente involucran un predicado construido, la comparación, si no que también la operación de suma. Sin embargo, se pueden trabajar como submetas, consideradas como una submeta EDB *sum(U,W,X)*, significando que el tercer argumento es la suma de los dos primeros. Entonces, se requiere que los dos primeros argumentos sean limitados cuando se accesa la relación para el predicado *sum*.

Estas reglas no son linealmente derechas ni linealmente izquierdas, ni tampoco linealmente contables para cualquier adorno de *en*, excepto *fff*, en último caso las reglas son linealmente a la izquierda ya que no existen argumentos limitados. Sin embargo, una transformación de conjuntos mágicos puede ayudar para una búsqueda típica $en(i_0,X,Y)$, es decir expandir la celda $i_0$. La idea esencial es que después de la transformación de conjuntos mágicos, el predicado mágico *m_on(J)*, procesa todas la celas *J* que están directa o indirectamente constituidas de $i_0$. Entonces se ve que se necesita encontrar, la porción de la relación *conjunto*, los puntos con valor 1. Finalmente se trasladan recursivamente esos puntos al sistema de coordenas de las celdas padres, hasta que eventualmente cada uno es trasladado al sistema de coordenadas de $i_0$.

**Ejemplo 9.2:** una operación importante en la optimización de compilación es el análisis de flujo de datos, en donde un programa se divide en bloques básicos (segmentos de código que se ingresan en la cima y se toman de la cola), e información con relación a las variables de los programas se propaga entre bloques. Un ejemplo simple es usar la definición de encadenamiento, donde lo que se desea es saber, para algún uso *U* de una variable *X* ( *U* puede ser A:=X+Y, por ejemplo), en que sentencia *X* pudo haber sido definida la última vez cuando el control del programa alcanza *U*. Así si se descubre que pudo haber sido definida previamente sólo en una asignación o asignaciones de la forma X:= 2, Entonces se sabe que *X* tiene el valor de 2 en *U*, y, se puede reemplazar *U* por la simple asignación *A:= Y +2*. La figura 9.1 ilustra los requerimientos para una definición D de una variables *X* para alcanzar un uso *U* de *X*.

Se necesita saber que *D* es "generada" por el bloque $B_1$, significando que *D* aparece en ese bloque, y que después de *D* no existe asignación alguna en $B_1$ que asigne un nuevo valor a la misma variable que asigna *D*. Cuando *D* es generada por $B_1$ también se dice que *D* "sale" de $B_1$ .

Entonces, se necesita encontrar un camino a través de los bloques, tal que *D* no sea "eliminado", significando que no existe asignación a *X* en ningún

bloque del camino. Si ese camino alcanza el bloque $B_n$ se dice que $D$ "entra" al bloque $B_n$. Si, además, $B_n$ tiene el uso de $U$ de $X$, y ese uso es "desmascarado", significa que antes de $U$ en el bloque $B_n$, no existe alguna definición de $X$, entonces, se dice que $D$ alcanza a $U$. La figura 9.2 muestra las reglas que expresan la notación de una definición para alcanzar una

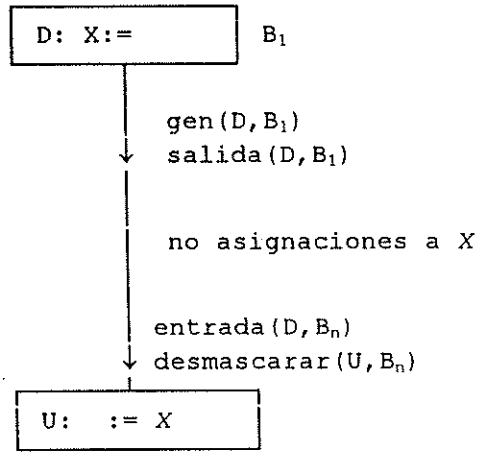



**Figura 9.1 Camino para la definición de D para alcanzar el uso de U.**

```
r1: borra(D,B) :- dvar(D,X) & ¬elimina(X,B).
r2: salida(D,B) :- genera(D,B_.
r3: salida(D,B) :- entrada(D,B) & borra(D,B).
r4: entrada(D,B) :- salida(D,C) & sucesor(C,B).
r5: alcanza(D,U) :- entrada(D,B) & desmascara(U,B) &
                    dvar(D,X) & uvar(U,X).
```

**Figura 9.2 Reglas para la definición de alcance.**

variable en uso. Los predicados EDB son:

1. genera(D,B), significa que la definición $D$ es generada por el bloque $B$.
2. desmascara(U,B), significa que $U$ se desmascara en el bloque $B$.
3. elimina(X,B), significa que el bloque $B$ elimina las definiciones con la variable $X$, es decir que $B$ tiene su definición de $X$.
4. dvar(D,X), significa que la definición de $D$ define la variable $X$.
5. uvar(U,X), significa que usar $U$ es usar la variable $X$.
6. sucesor(C,B), significa que el bloque $B$ es un sucesor del bloque $C$.

los predicados IDB son

7. entrada(D,B), significa que la definición $D$ está en el bloque $B$.
8. salida(D,B), significa que la definición D sale del bloque $B$.
9. borra(D,B), significa que la variable definida por la definición $D$ no está elimina por el bloque $B$.
10. alcanza(D,U), significa que la definición $D$ alcanza a $U$.

Se debe notar el uso inseguro de la negación en la regla $r_1$, donde la variable *B* aparece sólo en la cabeza y en la submeta negada y así no se limita a un conjunto finito. En la práctica, no existe problema, ya que cuando se trata ¬*elimina* como si fuera un predicado EBD, algunas veces llamado *trans*, donde *trans(X,B)* significa que el bloque *B* transmite, es decir, que no elimina la variable *X*. Ya que esto generalmente requiere menos espacio para almacenar *elimina*, en lugar de almacenar *bloque(B)*, una relación EDB adicional listando los bloques, simplemente se agrega la submeta *bloque(B)* a $r_1$, como

```
borra(D,B) :- dvar(D,X) & bloque(B) & ¬elimina(X,B).
```

De esta manera, se evita el tratar de inferir *borra(D,B)* para los bloques no existentes.

Se puede entender más fácilmente la recursión si se combinan las dos reglas recursivas $r_3$ y $r_4$, en una regla. Se hace esto, sustituyendo el cuerpo de la regla $r_4$ por la submeta *entrada(D,B)*, para crear una nueva regla, que se debe llamar $r_6$. Técnicamente, se debe unificar la cabeza de $r_4$ con esa submeta, pero, aquí el unificador más general es la función identidad.

```
r2: salida(D,b) :- genera(D,B).
r6: salida(D,B) :- salida(D,C) & sucesor(C,B) & borra(D,B).
```

Ahora se puede eliminar $r_3$, aunque se necesite $r_4$, que ahora no es recursiva, así que se puede procesar la relación *entrada* y usarla en la regla $r_5$.

Si la búsqueda es $salida^{bf}$, es decir, una definición dada *D* cuyos bloques salen de *D*, entonces, esta recursividad es linealmente izquierda. Es decir que la mejor forma de resolver $r_2$ y $r_6$ es explorar el grafo definido por la relación EDB *sucesor*, iniciando en el bloque que contiene a *D*, pero no siguiendo un arco sucesor a cualquier bloque que elimine la variable de *D*.

Desafortunadamente, se debe hacer una búsqueda de la forma $salida^{fb}$, es decir, para un bloque dado *B*, encontrar todas las definiciones que salen de *B*. Ahora $r_6$ no es lineal ni por la izquierda ni por la derecha. En particular no es lineal por la derecha porque la variable *D* compartida por el argumento libre de la cabeza y la submeta recursiva, también aparece en otra submeta, *borra(D,B)*. Se puede usar una transformación de conjuntos mágicos para acelerar la evaluación de $salida^{fb}$. El efecto de esta transformación es restringir el procesamiento de aquellos bloques que pueden alcanzar a *B* en el diagrama de flujo. Desafortunadamente, ese conjunto es frecuentemente una porción grande del diagrama de flujo.

### 9.2 El sistema NAIL.

NAIL (Not Another Implementation of Logic) es un sistema experimental basado en la lógica desarrollado por un grupo de estudiantes de Stanford. El principal objetivo del proyecto es estudiar la optimización de búsquedas. Como un resultado, la arquitectura, que se muestra en la figura 9.3 hace uso de un sistema de base de datos de SQL para manejar las propiedades estándar de un DBMS, así como el mantenimiento indexado. Indudablemente, esa no es la manera apropiada de manejar el acceso de una base de datos y un sistema de base de batos más integrado se necesita en un sistema comercial.

### 9.2.1 NAIL, código fuente.

Las reglas de la lógica son escritas en la notación presentada en este trabajo de tesis, pero, con la palabra reservada *not* en lugar de ¬. Las reglas deben ser seguras y la negación debe ser estratificada; es decir que no se permite la recursividad involucrando la negación. Un programa fuente se compila en tablas internas, involucrando algunas modificaciones a las reglas, que no se mostrará aquí. Lo más importante, los predicados son particionados en SCC (Strongly connected components), componentes fuertemente conectados, Un SCC es:

1. un predicado simple no recursivo (IDB o EDB);
2. un conjunto mínimo de predicados recursivos mutuos.

**Ejemplo 9.3:** el programa de reglas lógicas de la figura 9.2 es, esencialmente, código fuente NAIL, aunque $r_1$ debe ser reescrita, para evitar el uso inseguro de la negación, por ejemplo como

```
borra(D,B) :- dvar(D,X) & bloque(B) & not elimina(X,B).
```

Cada uno de los predicados IDB o EDB excepto *entrada* y *salida* están en sus propios SCC. Se coloca a *entrada* y *salida* en un SCC ya que son mutuamente recursivos.

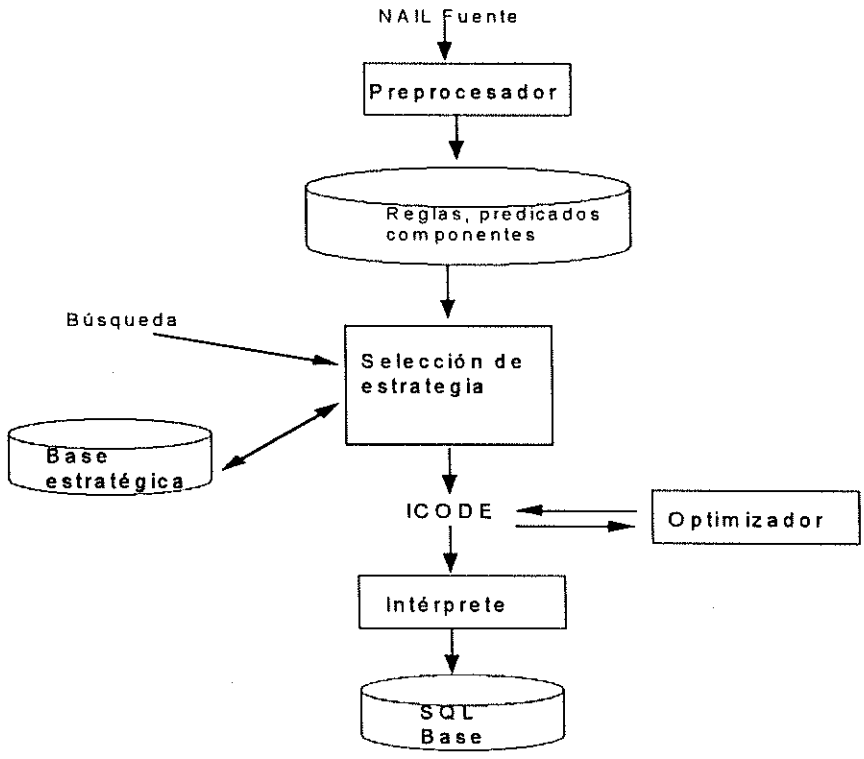



**Figura 9.3 Arquitectura de NAIL.**

Una vez que las reglas están compiladas, el sistema está listo para aceptar búsquedas. Una búsqueda es un predicado, con alguno de sus argumentos

constantes y otros variables. La respuesta de una búsqueda es el conjunto de tuplas para ese predicado que iguala las constantes. Para cada búsqueda, el sistema selecciona una estrategia de evaluación, que es una opción de algoritmos para la evaluación de una colección de metas ordenadas; esta colección incluye la búsqueda y otras metas determinadas por el sistema que son necesarias para la evaluación de la búsqueda.

### 9.2.2 Estrategia de Selección.

El SCC es la unidad por la cual se hace la decisión considerando cómo una pieza de la lógica será manejada. El componente de la selección de estrategia del sistema NAIL está diseñado para tomar una submeta adornada $p^{\alpha}$ y escojer una evaluación para las partes relevantes de la relación para $p$, dada una relación que contiene las tuplas de cualquier argumento $\alpha$ limitado. Para hacer esto se necesita ejecutar un algoritmo de selección de estrategia para resolver algunos problemas de bajo nivel, que están representados por las metas adornadas $q^{\beta}$, donde $q$ es un predicado en un SCC sobre el cual el SCC de $p$ depende. Es decir las reglas para los predicados en el SCC de $p$ normalmente tienen submetas cuyos predicados están fuera del SCC de $p$. Estos predicados deben ser de menor nivel que $p$, ya que si dependen de $p$, ellos serán mutuamente recursivos con $p$ y así se convierten en el mismo SCC de $p$.

Dada la meta adornada $p^{\alpha}$, el algoritmo de estrategia de selección inicia construyendo un grafo de reglas y metas para el SCC de $p$. Todos los predicados que no están en el SCC de $p$ se tratan como si fueran submetas EDB. Si se tiene éxito en la construcción del grafo de reglas y metas, se examinan todas las metas externas adornadas, que son aquellas cuyos predicados están fuera del SCC de $p$. Cada uno debe tener una estrategia seleccionada por otra aplicación del algoritmo de selección de estrategia. Ya que las metas externas son de un nivel más bajo que $p$, eventualmente se debe alcanzar un punto donde solamente se tengan metas EDB adornadas y metas aritméticas construidas para manejar. Por ejemplo se puede evaluar $=^{bf}$ (dado un conjunto de $\{a_1,\ldots,a_n\}$ de constantes como un encadenamiento, retornando un conjunto de pares $\{(a_1,a_1),\ldots,(a_n,a_n)\}$) pero no se puede evaluar $<_{bf}$, porque entonces se produce un conjunto infinito de pares.

**Ejemplo 9.4:** en la figura 9.4, uno de los posibles grafos de reglas y metas que deben ser construidos para la meta adornada $entrada^{bb}$ . Existen algunos grafos diferentes que deberían ser construidos , porque NAIL selecciona el orden de las submetas por la heurística:

1. tomar las submeta restante con el número más grande de argumentos limitados;
2. romper ataduras tomando la submeta con el menor número de argumentos libres;
3. si existen ataduras, romperlas arbitrariamente.

En este caso, hay algunos puntos donde las ataduras deben romperse arbitrariamente. Por ejemplo, en $r_4$, iniciando con $entrada^{bb}$, donde se toman los encadenamientos para ambas variables *B* y *D* que aparecen en la cabeza. Así, ambas submetas, *salida(D,C)* y *sucesor(C,B)*, tienen un argumento limitado y un argumento libre. Primero se selecciona $salida^{bf}$, seguido de $sucesor^{bb}$, pero se puede tener $sucesor^{fb}$ seguido de $salida^{bb}$. Dependiendo de los datos, uno o el otro orden pueden producir una evaluación más eficiente que la otra.

Asumiendo que NAIL ha construido el grafo de reglas y metas de la figura 9.4, se observa lo siguiente. La meta adornada $entrada^{bb}$ ha causado la

consideración de las metas adornadas internas *entrada*$^{bf}$ y *salida*$^{bf}$; y, también causa la necesidad de las soluciones para las metas externas adornadas, *genera*$^{bf}$, *sucesor*$^{bb}$, *sucesor*$^{bf}$ y *borra*$^{bb}$. Ya que todas estas metas son EBD, no hay razón para esperar que alguna de ellas falle. Como un resultado, *entrada*$^{bb}$ es exitosa . La información debe ser esencial para permitir una meta adornada de un SCC mayor.

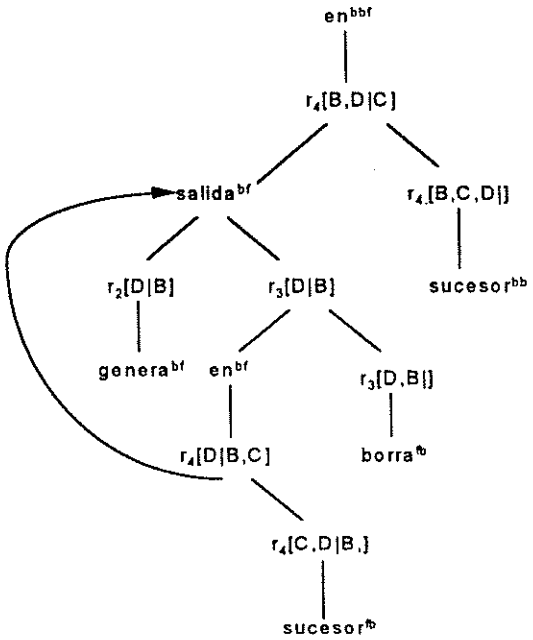



**Figura 9.4 Grafo de reglas y metas para el ejemplo de análisis de flujo de datos.**

Ahora, supóngase que por alguna razón, *genera*$^{bf}$ falla. Esto indicaría que el grafo de reglas y metas de la figura 9.4 no es aceptable. Se está en la capacidad de encontrar otro grafo de reglas y metas para *entrada*$^{bb}$, uno que evite la submeta *genera*$^{bf}$. Por ejemplo, se puede ordenar la submeta *salida* de $r_4$ de segundo lugar, en lugar del primero. Si se encuentra un grafo de reglas y metas aceptable , entonces, *entrada*$^{bb}$ es exitosa; de otra manera *entrada*$^{bb}$ falla.

### 9.2.3 Captura de reglas.

Las decisiones que consideren qué estrategia usar, para satisfacer una meta adornada, se hacen por captura de reglas. Cada captura corresponde informalmente a una idea para la evaluación de reglas de la lógica, tal como la recursión lineal izquierda, conjuntos mágicos o buscar una relación en la base de datos. Una captura de reglas consiste en dos piezas de código.

1. Un algoritmo *test* decide cuando una idea particular es aplicable a la meta adornada que se tiene. El *test* puede construir un grafo de reglas y metas, tal vez uno forzado por la necesidad de la idea informal, como es el caso para la recursividad izquierda o derecha, donde la meta recursiva debe aparecer en una posición particular con un adorno particular.

2. Un algoritmo de *ejecución*. Este algoritmo genera código intermedio para procesar la tuplas que encajan a la meta, dando una relación de encadenamiento para los argumentos limitados de esa meta.

La opción de capturar la regla para una meta adornada $p^a$ es un proceso de dos fases. Primero la clase a la que corresponde el SCC de $p$. El conjunto de clases es fácilmente ajustable con el sistema NAIL, pero cada SCC debe pertenecer exactamente a una clase, el conjunto inicial de clases es:

1. Predicado EDB.
2. Predicados construidos (comparación aritmética).
3. Predicados de Datalog no recursivos.
4. Predicados que no son Datalog no recursivos, es decir , un predicado no recursivo una o más de cuyas reglas tienen funciones de símbolos.
5. Datalog recursiva lineal.
6. No Datalog recursiva lineal.
7. Datalog recursiva no lineal.
8. No Datalog recursiva no lineal.

Para cada clase se lista una o más reglas de captura. La porción del test de estas reglas se ejecuta hasta que se tiene éxito. Por ejemplo, la clase recursiva lineal de Datalog debe tener, en orden, reglas de captura para las estrategias

i) Lineal izquierda.
ii) Lineal derecha.
iii) Lineal izquierda y derecha.
iv) Lineal contable.
v) Conjunto mágicos.

Sin embargo, algunas condiciones, aparentemente no del grafo de reglas y metas, son necesarias para aplicar algunos de estos métodos. Los métodos (i) a (iii) requieren que el conjunto de encadenamiento sea único. Así que sólo pueden ser aplicados seguramente a la búsqueda inicial y en algunos casos especiales; en general no se puede decir, en el tiempo de procesar la búsqueda, qué relaciones de encadenamiento serán únicas. El método (iv) requiere que ciertos predicados tengan relaciones acíclicas. Para proporcionar las declaraciones adicionales necesarias para hacer tales decisiones, la fuente de NAIL tiene un predicado construido *extraHelp*, que permite transmitir tales pistas al algoritmo de estrategia de selección.

### 9.2.4 Base de datos de estratégicas.

Cuando NAIL descubre una estrategia para una meta adornada, almacena estas decisiones en una base de datos de estrategias. Entonces, la próxima vez que se encuentre la misma meta adornada, aunque tenga una relación de encadenamiento diferente, no se tiene que reconsiderar la captura de reglas y el grafo de reglas y metas; en lugar de esto, se toma la estrategia propia en la base de datos.

Lo que se almacena para una captura exitosa de $p^a$ es

1. la meta adornada externa y, si se usa, el grafo de reglas y metas para la SCC de $p^a$;
2. la captura de reglas que fue exitosa.

Ya que se tuvo éxito en la captura de $p^a$, se sabe que también se tiene éxito con todas las metas adornadas externas para este SCC. Así se sabe que

la estrategia exitosa para cada una de las metas externas será almacenada en la base de datos de estrategias y se puede cargar cuando se genere código intermedio para esos SCC.

Para evitar búsquedas repetidas sin objeto, también se almacenan datos negativos, indicando que no se encontró una captura de reglas capaz de manejar cierta meta adornada $p^{\alpha}$. Antes de atender la captura de cualquier $p^{\beta}$ se debe de buscar en la base de datos de estrategias, y si ß no es más limitada que $\alpha$, es decir, $\alpha$ tiene *b* donde ß tiene *b*, se debe concluir, sin hacer más búsquedas, que $p^{\beta}$ no puede ser capturada. Lo racional es que un argumento limitado no debe hacer más difícil la respuesta de una búsqueda. En la peor de las búsquedas, se puede usar un método donde se pretende que los argumentos limitados sean libres y entonces, en el último paso, ejecutar una selección sobre ese argumento.

Si se encuentra que $p^{fb}$ puede ser capturada por alguna estrategia, entonces, se espera que $p^{bb}$ también pueda ser capturada. Sin embargo, puede existir una estrategia eficiente más simple que trabaje para $p^{bb}$ pero no para $p^{fb}$. Si se omite hacer una búsqueda completa de estrategia para $p^{bb}$, sólo porque se encontró una estrategia para $p^{fb}$ en la base de datos de estrategias, entonces, faltará esta estrategia más eficiente. Sin embargo, podría no encontrarse una mejor estrategia para $p^{bb}$, entonces, se debe ingresar la misma estrategia que para $p^{fb}$ en la base de datos, esto evita la búsqueda de una mejor estrategia que no existe.

### 9.2.5 Código intermedio NAIL.

Como se indicó en la figura 9.3, el resultado de una estrategia exitosa para responder una búsqueda es un programa en un lenguaje llamado ICODE. Este lenguaje consiste de sentencias para cada operación del álgebra relacional, selección, proyección, producto, unión, diferencia, etc. Todo lo que se refiere a la diferencia y la unión puede ser ejecutado por las sentencias de ICODE en la forma de las reglas de Prolog. Sin embargo, a diferencia de las sentencias de Prolog, una regla en ICODE tiene un significado operacional en lugar de tener un significado declarativo; cuando se hace una sentencia de éstas, la relación de la cabeza se evalúa en términos de las relaciones del cuerpo, y la nueva relación es almacenada en la base de datos.

Para permitir la secuencia de sentencias, ICODE tiene suficientes controles de flujo de control. Las variables de ICODE pueden ser tratadas como arreglos o pilas de relaciones.

Las sentencias de ICODE permiten funciones de símbolos y operadores aritméticos para enteros. Como un resultado, existe la posibilidad de que relaciones construidas contengan valores que no son elementales, es decir términos complejos. Ya que todos los datos son almacenados en una base de datos de SQL, todos los términos complejos deben ser descritos como tuplas de las relaciones, NAIL usa una relación simple llamada CONS, en donde se almacenan los nodos del árbol de análisis de los términos complejos. Por ejemplo, para almacenar el término *f(a,b,g(c))*, debe haber una tupla representando *f* y su primer argumento *a*. Una segunda tupla debe representar los dos siguientes argumentos de *f*, que son, *b* y *g(c)*. La estructura del subtérmino *g(c)* debe ser representado por una tercera tupla que contenga el símbolo *g* y el primer argumento *c*.

**Ejemplo 9.5** : si se toman las reglas usuales para ancestros,

```
r1 : anc(X,Y) :- par(X,Y).
r2 : anc(X,Y) :- par(X,Z) & anc(Z,Y).
```

con la búsqueda *anc(joe,Y)*. NAIL usa la transformación lineal derecha para evaluar la búsqueda. Es decir, NAIL actúa como si las reglas hubieran sido

```
m_anc(joe).
m_anc(Z) :- m_anc(X) & par(X,Z).
a_anc(Y) :- m_anc(X) & par(X,Y).
anc(joe,Y) :- a_anc(Y).
```

**Figura 9.5 Transformación lineal derecha sobre las reglas de ancestro.**

transformadas, como se muestra en la figura 9.5. Estas reglas pueden ser transformadas al programa de ICODE que se muestra en la figura 9.6.

```
(1)    relation(par(child,parent));
(2)    initial(dma,(joe));
(3)    makeEmpty(ma);
(4)label(beginLoop)
(5)    union(ma,dma,ma);
(6)    ndma(Z) :- dma(X), par(X,Z);
(7)    diff(ndma,ma,dma);
(8)    ifNotEmpty(beginloop,dma);
(9)    aa(Y) :- ma(X), par(X,Y);
(10)   anc(Joe,Y) :- aa(Y);
```

**Figura 9.6 ICODE para la evaluación de las reglas de ancestro.**

Las variables de ICODE *ma* y *aa* corresponden a los predicados *m_anc* y *a_anc*, respectivamente. La variable *dma*, o delta del ancestro mágico, es el conjunto de tuplas agregadas a *m_anc* en el ciclo más reciente. Finalmente, *ndma*, o nuevo delta del ancestro mágico, es una temporal que ayuda a procesar *dma*.

La sentencia (1) de la figura 9.6 declara que la relación *par* es una relación permanente en la base de datos, con los atributos nombrados *child* y *parent*. Otras relaciones no declaradas, se crean cuando se necesitan y se borran de la base de datos antes de que termine el programa de ICODE.

La sentencia (2) utiliza *dma* para *{Joe}*, y la sentencia (3) inicializa *ma*. En la sentencia (5), el inicio del ciclo, el *dma* actual o las nuevas tuplas de *ma*, son insertadas en *ma*, por ejemplo, en el primer ciclo *ma* se iguala a *{Joe}*. En general, *unión(A,B,C)* significa *C:= A U B*.

La sentencia (6) procesa *ndma* para igualarla al conjunto de padres de individuos que se encuentran en *dma*; estos deberían ser individuos que no se han encontrado en ciclos anteriores, pero que ahora se han convertido en alcanzables ya que uno de sus hijos fue descubierto y por lo tanto se ingreso

en *dma*, en un ciclo anterior. En la línea (7) se borran todos los individuos que ya han sido alcanzados con anterioridad y los individuos restantes, los individuos que no se conocían antes, se convierten en nuevos valores de *dma*. En general, *diff(A,B,C)* significa *C:= A - B*.

La sentencia (8) produce un salto de regreso al principio del ciclo si *dma* no está vacía, es decir, si se tienen más individuos para considerar. Cuando se ha encontrado el conjunto mágico completo, consistente de *Joe* y sus ancestros, se llega a las sentencias (9) y (10) que implementan las dos últimas reglas de la figura 9.5 para procesar la respuesta.



PROPIEDAD DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Biblioteca Central

## CONCLUSIONES

1. Los sistemas basados en la lógica son sistemas con lenguajes declarativos y lenguajes basados en la lógica.

2. Existe una diferencia fundamental entre la aproximación orientada a objetos y la aproximación a la lógica, para diseñar un lenguaje integrado, el último es inherentemente declarativo y el primero no lo es.

3. El término declarativo de los lenguajes de búsquedas tiene una influencia significante sobre la arquitectura del sistema completo de la base de datos. Los siguientes puntos resumen las diferencias observadas entre los sistemas con lenguajes declarativos y procedurales, aunque se deba enfatizar que estas aseveraciones son generalidades que pueden ser contradichas por otros avances en la tecnología de las bases de datos.

a. Los usuarios prefieren lenguajes declarativos

b. Los lenguajes declarativos son más duros de implementar que los lenguajes procedurales porque los lenguajes declarativos requieren de una extensiva optimización por el sistema.

c. Aparece que la orientación a objetos y lenguajes declarativos son incompatibles.

4. SQL y lenguajes similares, para la manipulación de datos, no tienen el suficiente poder para procesar cerraduras transitivas, tal como el manejo jerárquico, las reglas lógicas que usan las funciones de símbolos tienen todo el poder de una máquina de turing; pueden expresar cualquier computación que puede escribirse en un lenguaje de programación convencional. Las mismas reglas de la lógica sin las funciones de símbolos tienen el poder de expresar más computaciones que un lenguaje de manipulación de datos, convencional.

5. La lógica de predicados es una forma de representar "conocimiento" y de expresar operaciones sobre relaciones.

6. Las dos formas restringidas de Datalog, llamadas dominio de cálculo relacional y cálculo relacional de tupla, son equivalentes en poder al álgebra relacional y forman las bases de la mayoría de los lenguajes de búsquedas comerciales en los sistemas relaciónales.

7. La comunidad de la inteligencia artificial, conoce que existen ventajas del procesamiento descendente y ascendente de las reglas. Una nueva técnica, conjuntos mágicos, permite obtener las ventajas de ambos procesos, con las desventajas de ninguno de los dos. Esto hace que los conjuntos mágicos se muestren superiores en cualquier aplicación.

8. En Guatemala la aplicación de un sistema basado en la lógica se podría usar en aplicaciones gubernamentales, donde se trabaja con montos masivos de datos y la necesidad de una facilidad de búsqueda más poderosa que los lenguajes de búsqueda típicos como SQL, es más que obvia. Lastimosamente, no se conoce de un un manejador de base de datos basado en la lógica comercial.

## RECOMENDACIONES

1. La tendencia actual en los sistemas de basés de datos, son los sistemas orientados a objetos, que soportan tipos abstractos de datos, de éstos se dice que no están orientados a valores o a registros. Esto pude hacer pensar que los sistemas basados en la lógica no sean una tendencia de actualidad ya que de igual manera que el modelo de datos relacional, los sistemas basados en la lógica están orientados a valores.

Se puede suponer por naturaleza que la orientación a objetos es preferible a la orientación a valores. Sin embargo, se puede falsear la identidad de un objeto en un sistema orientado a valores con el uso de un campo que sirva como un número serial para objetos.

Además, en favor de los sistemas orientados a valores, la identidad de objetos no se lleva de la mano con la capacidad declarativa de estos sistemas.

Se recomienda, por lo tanto, que la orientación, a valores, de los sistemas basados en la lógica, no sea motivo para creer que no son una tendencia de la actualidad.

2. Se recomienda que cuando se encuentre el término "conocimiento" en un producto comercial, no se considere que su ejecución y funcionalidad debe ser mejor que cualquier producto similar que no muestre tales cualidades. Ya que existe una gran diferencia entre los sistemas de conocimiento y los sistemas de administración de base de conocimientos es decir sistemas basados en la lógica.

3. Se recomienda a todas la personas que pretendan utilizar este material, que profundicen en los conceptos de bases de datos tradicionales así como de la lógica formal, antes de estudiar a la lógica como un modelo de datos.

4. Se recomienda el presente trabajo como material de complemento para los estudiantes de la carrera de Ciencias y Sistemas de la Computación, en especial a los interesados en el área de las bases de datos.

5. Se recomienda este material como una recopilación de conceptos básicos. Un estudio mucho más profundo de los algoritmos de búsquedas de otros sistemas que no están basados en la lógica relacionados con los que en este trabajo se presentan. Podría generar un grupo de algoritmos para la optimización de consultas para los sistemas basados en la lógica.

6. Se recomienda usar este trabajo como base para la construcción de un sistema de este tipo donde se puedan aplicar los conceptos expuestos.

## BIBLIOGRAFÍA


Andriole, Stephen J.
Applications in Artificial Intelligence. USA: Editorial Petrocelli Books, 1987. Inc.

Campbell,J A.
Implementations of Prolog. s.l.i. USA., s.f.

Clocksin, William F. et. al.
Programming in Prolog. USA: Editorial Springer-Verlang, 1985.

Date, C. J.
Sistemas de Bases de Datos. 5a. edición. USA: Editorial Addison Wesley Iberoamericana, 1990.

Fadden, Fred R. et. al.
Modern Database Management. USA: Editorial Benjamin Cummings, 1992.

Frost, Richard
Bases de Datos y Sistemas Expertos. Ingeniería del Conocimiento. Madrid: Editorial Díaz de Santos, S. A., 1989. Traducción de José Calle Gaugliere.

Hayes-Roth ,Frederick et. al.
Building Expert Systems. Canadá: Addison Wesley Publishing Company,Inc. 1993.

Kamran, Persaye et. al.
Intelligent Databases. USA: Editorial Wiley, 1989.

Kolman, Bernard y Robert C. Busby
Estructuras de Matemáticas discretas para la computación. México: Editorial Prentice Hall Hisponoamericana, S.A., 1986.

Korth, Henry Y Abraham Silberschatz
Fundamentos de Bases de Datos. 2a edición. México: Editorial McGraw Hill Interamericana de México, S. A., 1992.

Kowalski, Robert
Logic As a Computer Language. England: Department of computing Imperial College, s.f.

Randall, Davis
Knwoledge-Bases Systems In Artificial Intelligence. USA: Editorial McGraw Hill International Book Company, 1982.

Tarnlud, K. L. Clark
Logic Programming. USA : Editorial Academic Press Inc.,1989.

Ullman, Jeffrey
Database and Knwoledge Base Systems. 2 vols.; USA: Editorial Computer Science Press, 1989.