UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

Una Solución Eficiente al Problema de Reescritura de Consultas

Trabajo de Grado presentado a la Universidad Simón Bolívar por
Yolifé Arvelo Noriega

Como Requisito Parcial para Optar al Grado de
Magíster en Ciencias de la Computación

Realizado con la tutoría de los profesores
María Esther Vidal
Blai Bonet

Noviembre, 2006

UNIVERSIDAD SIMÓN BOLÍVAR
Decanato de Estudios de Postgrado
Maestría en Ciencias de la Computación

UNA SOLUCIÓN EFICIENTE AL PROBLEMA DE REESCRITURA DE CONSULTAS

Este Trabajo de Grado ha sido aprobado en nombre de la Universidad Simón Bolívar por el siguiente jurado examinador:

Presidente
Elías Tahhan

Miembro Externo
Claudia León
Universidad Central de Venezuela

Miembro Externo
Ramón Pino
Universidad de los Andes

Miembro Interno-Tutor
María Esther Vidal

Miembro Interno-Tutor
Blai Bonet

Fecha: 27/11/2006

A mis padres Julio y Yolanda
A mis tutores Blai y María Esther

Resúmen

El problema de responder consultas usando vistas consiste en encontrar reescrituras de una consulta usando un conjunto de vistas, en lugar de acceder directamente las relaciones de la base de datos. Este problema ha recibido gran atención en el área de integración de datos debido al surgimiento de nuevas infraestructuras que requieren métodos eficientes para evaluar consultas. En este contexto, no se puede asumir que la información de las vistas es completa, por lo cual es de interés encontrar reescrituras que produzcan la mayor cantidad de respuestas a partir de las vistas, es decir, las reescrituras maximalmente contenidas.

En general, el problema es intratable, ya que pueden existir un número exponencial de maneras de combinar las vistas para responder la consulta. Existen diversos algoritmos que abordan el problema de manera eficiente, entre ellos se encuentran: *Bucket, Inverse Rules*, y *Minicon*. El algoritmo más conocido, *Minicon*, intenta reducir el espacio de búsqueda considerando sólo las vistas que puedan cubrir algún subobjetivo de la consulta. Luego, realiza todas las posibles combinaciones de vistas relevantes para formar las reescrituras. Aunque esta solución es más eficiente que los otros algoritmos, no escala bien cuando se aumenta el tamaño de la consulta y las vistas o el número de vistas disponibles.

En este trabajo se propone una solución al problema que permite enumerar las reescrituras, de manera eficiente. En la solución propuesta, las condiciones a ser satisfechas por las reescrituras para que estén contenidas en la consulta son representadas en lógica proposicional. En consecuencia, los modelos de esta teoría están en correspondencia con las reescrituras. Para identificar los modelos, la teoría es compilada en una forma normal, como d-DNNF *Deterministic decomposable negation normal form*, que permite enumerar y contar modelos en tiempo polinomial, en el tamaño de la teoría compilada. Aunque, realizar la transformación de la teoría es intratable en el peor caso, no es necesariamente así en promedio.

El presente trabajo muestra un mejora significativa del tiempo de ejecución en comparación con el algoritmo de *Minicon*. Adicionalmente, la reducción del problema a lógica proposicional no sólo permite la implementación de algoritmos más rápidos y sencillos, sino que permite pensar en soluciones más generales al problema.

Palabras clave: Integración de datos, Reescritura de consultas, Compilación de conocimiento.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

El problema de responder consultas usando vistas [GMPQ$^+$97, CGLV03, CGLV00, Hal00, Hal01, Len02, MLF00, PH01] consiste en encontrar métodos eficientes para reescribir una consulta usando un conjunto de vistas, en lugar de acceder directamente las relaciones de la base de datos. Informalmente, el problema puede describirse de la siguiente manera: dada una consulta Q y un conjunto de definiciones de vistas $\mathcal{V} = V_0, V_1, \cdots, V_n$ sobre el esquema de la base de datos, se desea responder Q usando la información que producen las vistas.

El problema ha sido estudiado desde hace algún tiempo, debido a que existen muchas aplicaciones en las cuales se puede presentar [GMPQ$^+$97, CKPS95, DG97b, Hal01, Len02, LRO96a, LRU96, TS97, TSI96, YL87, ZCL$^+$00]. Un primer tipo de aplicación donde es posible encontrar el problema de reescrituras es la optimización de consultas [BDD$^+$98, CKPS95, TSI96]. En este contexto, es posible optimizar una consulta utilizando vistas previamente materializadas, si parte del trabajo necesario para responder la consulta ya ha sido realizado al materializar la vista. Pero para decidir qué vista utilizar es necesario obtener las posibles reescrituras de la consulta en términos de las vistas.

Otra aplicación, donde se puede encontrar este problema, es en sistemas integración de datos [DG97b, Hal01, Len02, LRO96a, LRU96]. Este tipo de sistemas proveen una interfaz uniforme para acceder a múltiples fuentes de datos autónomas y posiblemente heterogéneas. Para poder dar esta interfaz uniforme, el sistema define un esquema global sobre el cual se realizan las consultas y las fuentes de datos se definen como correspondencias entre este esquema y el esquema que las fuentes pueden responder.

Una manera de definir la correspondencia entre las fuentes y el esquema global, conocido en la literatura como *Local as view* [CGL01, Len02, LRO96c] y que ha sido adoptada por diferentes sistemas [DG97b, LRO96b], consiste en describir las fuentes de datos como vistas sobre el esquema

global. Con este enfoque, el problema de reformular una consulta sobre el esquema global en una consulta que puedan responder las fuentes de datos, se convierte en el problema de reescritura de consultas usando vistas.

Dada la diversidad de aplicaciones que puede tener este problema, se ha desarrollado mucho trabajo en el área, desde sus bases teóricas, hasta diseño de algoritmos e implementación de sistemas. Los algoritmos más conocidos en el área, *Bucket* [LRO96a, LRU96], *Minicon* [PH01], *Query rewriting* [Mit99], *Inverse rules* [DG97a, Qia96], atacan el problema de manera eficiente evitando, en lo posible, realizar combinaciones de vistas que no sean válidas. Pero si se aumenta el tamaño de las consultas y las vistas o el número de vistas disponibles, aumenta el número de posibles combinaciones válidas, por lo cual, estos algoritmos no escalan bien.

El objetivo de este trabajo es definir un algoritmo eficiente para resolver el problema, que pueda escalar si el número de vistas es grande o la consulta es compleja. La solución propuesta consiste en reformular el problema en una teoría de la lógica proposicional en la cual los modelos de la teoría estén en correspondencia con las reescrituras de la consulta usando las vistas. Finalmente, resolver el problema consistiría en enumerar los modelos de esta teoría. Resultados preliminares de este trabajo han sido reportados en [ABV06].

Con esta solución, se busca conseguir un método eficiente para resolver el problema de reescritura explotando el reciente desarrollo de técnicas del área de compilación de conocimiento. Estas técnicas consisten en transformar la teoría lógica en un lenguaje de compilación apropiado que permita hacer ciertas operaciones tratables, por ejemplo, la enumeración de modelos. Aunque, realizar la compilación de la teoría es intratable en el peor caso, no es necesariamente así en promedio.

Recientemente, se han desarrollado compiladores para diferentes lenguajes de compilación que son capaces de tratar con teorías de cientos de miles de variables y cláusulas. Su uso en el problema de reescrituras de consultas usando vistas puede contribuir a mejorar el desempeño de los algoritmos de reescritura en presencia de un gran número de vistas. Concretamente, en este trabajo se utiliza el lenguaje de compilación d-DNNF *Deterministic decomposable negation normal form* [Dar01], que permite enumerar y contar modelos en tiempo polinomial, en el tamaño de la teoría compilada.

Desde el punto de vista de complejidad, esta transformación no representa una mejoría con respecto a otros algoritmos. Debido a que el problema de conseguir una reescritura se reduce al problema de decidir si una asignación de variables de la teoría es satisfacible, lo cual sigue siendo un problema intratable [PS82]. Sin embargo, el estudio experimental, realizado en trabajo, muestra

una mejora significativa del tiempo de ejecución de nuestro enfoque con respecto al algoritmo más conocido en el área.

El objetivo general de este estudio es aplicar técnicas desarrolladas en el área de compilación de conocimiento para resolver el problema de reescritura de consultas usando vistas. Para poder cumplir con este objetivo, se plantearon los siguientes objetivos específicos:

1. Diseñar una teoría en lógica proposicional en la que todo modelo corresponda con una reescritura válida en el problema de reescrituras.
2. Construir una herramienta que genere la teoría lógica correspondiente al problema, obtenga los modelos de la misma y los convierta en las reescrituras.
3. Comparar experimentalmente el tiempo de ejecución del algoritmo más conocido para resolver el problema de reescritura de consultas con el tiempo de compilación de la teoría extendida correspondiente.

El presente trabajo se encuentra dividido en 7 capítulos, incluida la introducción. El capítulo 2 muestra, a través de ejemplos, el problema de reescritura de consultas usando vistas. El capítulo 3 introduce las definiciones necesarias para comprender el problema planteado. En el capítulo 4 se presentan los algoritmos existentes para reescrituras de consultas y se expone los aspectos más importantes del área de representación de conocimiento. El capítulo 5 se dedica a explicar como se diseñó la teoría lógica para representación de reescrituras. El capítulo 6 presenta un estudio experimental del desempeño del algoritmo desarrollado y finalmente, el capítulo 7 contiene las conclusiones y recomendaciones para trabajos futuros.

Capítulo 2

Motivación

Como ya se mencionó, existen diversas aplicaciones donde puede aparecer el problema de reescritura de consultas usando vistas, entre ellas: optimización de consultas usando vistas materializadas e integración de datos.

Sin embargo, la mayor parte del trabajo realizado en el área, se ha llevado a cabo debido a su aplicabilidad en sistemas de integración de datos [Hal01]. Es por eso que la sección 2.1, se dedica a describir y ejemplificar el problema de reescrituras en el contexto de integración de datos.

2.1. Integración de datos

El objetivo de los sistemas de integración de datos es dar al usuario una interfaz uniforme para acceder a un conjunto de fuentes de datos. Algunos ejemplos de aplicaciones de este tipo incluyen integración de fuentes de datos en organizaciones, consultas sobre múltiples fuentes de datos en el *Web*, integración de información científica distribuida, etc.

La figura 2.1 muestra la arquitectura de mediadores y traductores [BRV98, BGRV99, FLM98, GRVB98, TRV96, VR98, Vid00, VRG98, Wie97, ZBR$^+$00] comúnmente usada en sistemas de integración. En esta arquitectura, los mediadores ocultan la presencia de varias fuentes de datos mostrando un ente único para la realización de consultas. El mediador mantiene un esquema común que describe la información que almacenan las distintas fuentes, viéndolas como un conjunto.

Los traductores se encargan de recibir las consulta descrita en el esquema global y transformarla en el esquema que corresponde a la fuente de datos. Las fuentes de datos pueden ser desde

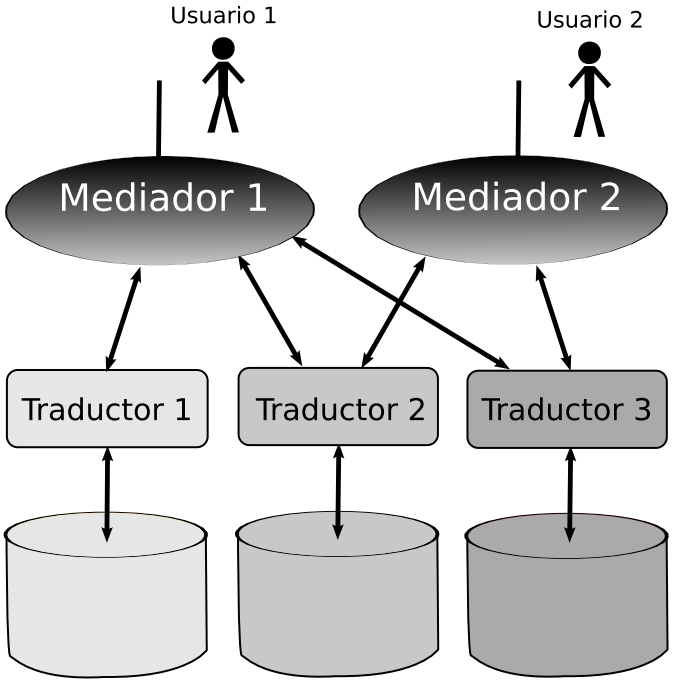

Figura 2.1: Arquitectura de mediadores y traductores.

bases de datos relacionales, que se encuentren dentro de una misma organización, hasta fuentes de datos diversas en internet.

El sistema de integración mantiene las relaciones o correspondencias que existen entre su esquema global y las fuentes de datos. Luego, usa estas correspondencias para acceder las fuentes necesarias para responder una determinada consulta sobre el esquema global.

Para especificar las correspondencias entre el esquema global y el esquema de las fuentes se han definido dos enfoques [CGL01, FMR+99, Len02, TRV98, Ull00]. El primer enfoque, *Global as view* o centrado en el esquema global, define el esquema global en términos de las fuentes de datos. Es decir, a cada elemento en el esquema global se asocia una vista sobre las fuentes de datos.

El segundo enfoque, *Local as view* o centrado en las fuentes de datos, define las fuentes de datos como vistas sobre el esquema global. Luego, la relación entre el esquema global y las fuentes de datos se establece por la definición de la información contenida en cada vista en términos del esquema global. Como consecuencia, el esquema global es independiente de las fuentes de datos.

Diferentes sistemas [LRO96b, DG97b, VRG98, ZRV+02], describen las correspondencias entre el esquema global y las fuentes de datos usando vistas. De esta manera es posible agregar nuevas fuentes de datos y se facilita la definición de restricciones sobre los datos que se encuentran en

las fuentes. El ejemplo 2.1 ilustra el problema de integración de datos, describiendo las fuentes de datos como vistas sobre el esquema global.

Ejemplo 2.1 (Integración de bases de datos de vuelos). Considere el esquema global, que se presenta en la tabla 2.1, para un sistema de vuelos entre diferentes ciudades.

Relación	Descripción
`flight`$(ICity, ECity)$	Existe un vuelo directo de la ciudad $ICity$ a la ciudad $ECity$
`uscity`$(City)$	La ciudad $City$ se encuentra en los Estados Unidos

Tabla 2.1: Esquema global para un sistema de vuelos

Dentro del sistema se encuentran tres bases de datos. La primera base de datos mantiene vuelos dentro de los Estados Unidos. La segunda base de datos mantiene vuelos directos entre dos ciudades. Finalmente, en la tercera base de datos se encuentran vuelos entre dos ciudades que hacen una parada en alguna ciudad. Las bases de datos se ilustran en la figura 2.2.

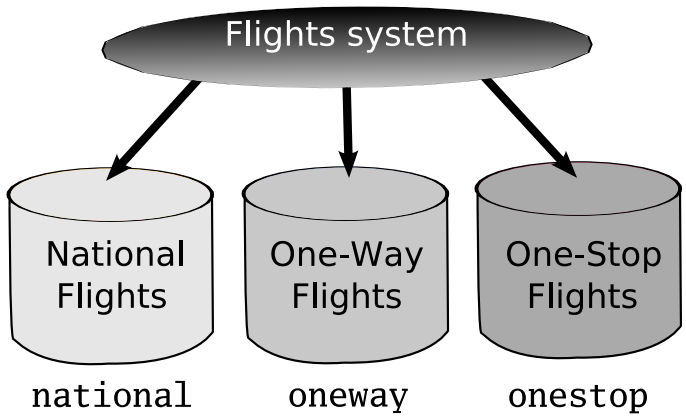

Figura 2.2: Bases de datos para el sistema de vuelos

Las descripciones de estas fuentes de datos, en DML, se presentan a continuación:

```
CREATE VIEW national
SELECT F.ICity , F.ECity
FROM flight F, uscity U1,
     uscity U2
WHERE F.ICity = U1.City
  AND F.ECity = U2.City
```

```
CREATE VIEW Oneway
SELECT F.ICity , F.ECity
FROM flight F
```

```
CREATE VIEW Onestop
SELECT F1.ICity , F2.ECity
FROM flight F1, flight F2
WHERE F2.ICity = F1.ECity
```

Suponga que el sistema necesita obtener todos los vuelos ida y vuelta entre dos ciudades. En términos del esquema global del sistema, esta consulta se puede expresar en SQL de la siguiente manera:

```
SELECT F1.ICity, F2.ECity
FROM flight F1, flight F2
WHERE F1.ICity = F2.ECity
  AND F2.ICity = F1.ECity
```

Debido a que la información en las fuentes de datos puede ser incompleta, el sistema de integración no puede garantizar que encuentra todas las respuestas posibles a la consulta. En su lugar, el sistema trata de obtener el conjunto maximal de respuestas disponibles a partir de las fuentes disponibles. En este caso, debe realizar todas las posibles combinaciones de vuelos que se puedan obtener de las bases de datos del sistema. Con lo cual, la respuesta a esta consulta es la unión de las respuestas producidas al combinar las bases de datos `oneway` y `national`. Esta reescritura se representa en SQL de la siguiente manera:

```
SELECT N1.ICity, N2.ECity
FROM national N1, national N2
WHERE N1.ICity = N2.ECity
  AND N2.ICity = N1.ECity
UNION
SELECT N1.ICity, N2.ECity
FROM national N, oneway O
WHERE N.ICity = O.ECity
  AND O.ICity = N.ECity
UNION
SELECT N1.ICity, N2.ECity
FROM national N, oneway O
WHERE O.ICity = N.ECity
  AND N.ICity = O.ECity
UNION
SELECT O1.ICity, O2.ECity
FROM oneway O1, oneway O2
WHERE O1.ICity = O2.ECity
  AND O2.ICity = O1.ECity
```

Obsérvese que la base de datos `onestop` no es capaz de responder la consulta ya que no produce como respuesta el atributo que representa la ciudad intermedia. Por lo tanto, considerar cualquier

combinación en la que forme parte la vista de `onestop`, al final se convertirá en una reescritura no válida. Cualquier algoritmo de reescritura eficiente debería ser capaz de notar este hecho desde el comienzo. En particular, los algoritmos de reescritura más recientes no tomarían en cuenta esta vista para construir las reescrituras [PH01, Mit99].

Es importante notar que en integración de datos se requiere que el tiempo en que se reescriba la consulta sea lo menor posible. El problema es que en muchos casos se requiere que este tipo de sistemas pueda ser capaz de tratar con muchas fuentes de datos y consultas de distinta complejidad, por lo cual el tiempo para obtener las reescrituras puede ser alto.

Por otro lado, calcular la respuesta a partir de la reescritura de la consulta es costoso, por lo cual es necesario encontrar reescrituras que sean minimales, es decir, si una vista es eliminada de la reescritura entonces ya no se satisface la consulta [Mit99].

Capítulo 3

Planteamiento del problema

En este capítulo se presentan las definiciones necesarias para comprender el problema de encontrar la reescritura de una consulta Q con respecto a un conjunto de vistas $\mathcal{V}$, en el caso en que las vistas y la consulta son consultas conjuntivas. En este trabajo no se consideran consultas con constantes o predicados aritméticos en el cuerpo, ni negación.

En la sección 3.1 se definen las consultas conjuntivas. Posteriormente, en la sección 3.2, se presenta la contención de consultas conjuntivas, como un concepto estrechamente relacionado con las reescrituras. El problema de reescritura de consultas se define formalmente en la sección 3.3. Finalmente, en las secciones 3.4 y 3.5 se presentan los tipos de reescrituras.

3.1. Consultas conjuntivas

El problema de reescritura de consultas, ha sido presentado comúnmente en la literatura, utilizando consultas conjuntivas. A continuación, se introducen formalmente las consultas conjuntivas.

Definición 3.1 (Consultas conjuntivas). Una consulta conjuntiva es de la forma:

$$q(\vec{X}) : -p_0(\vec{X}_0), p_1(\vec{X}_1), \ldots, p_m(\vec{X}_m)$$

donde $q, p_0, p_1, \ldots, p_m$ son predicados. Los átomos $p_0(\vec{X}_0), p_1(\vec{X}_1), \ldots, p_m(\vec{X}_m)$ son los subobjetivos en el cuerpo de la consulta y $p_0, p_1, \ldots, p_m$ se refieren a relaciones en el esquema de la base de datos.

Las tuplas $\vec{X}, \vec{X}_0, \vec{X}_1, \cdots, \vec{X}_m$ pueden contener variables o constantes. Se dice que una variable x es distinguible cuando aparece en la cabeza de la consulta y se denota como $x \in Dist(Q)$. Si la variable x no es distinguible, entonces se dice que es existencial y se denota como $x \in Exist(Q)$.

Todas las consultas son seguras, es decir, toda variable en la cabeza debe aparecer en el cuerpo, es decir, $\vec{X} \subseteq \vec{X}_0 \cup \vec{X}_1 \cup \cdots \cup \vec{X}_m$.

La respuesta de Q con respecto a una base de datos D se refiere al resultado de evaluar la consulta Q sobre D y se denota como $Q(D)$.

Ejemplo 3.1. Considere el sistema de vuelos presentado en el ejemplo 2.1.

La siguiente consulta busca vuelos ida y vuelta entre ciudades de los Estados Unidos:

$$Q(x, y) \ :- \ \texttt{flight}(x, y), \texttt{flight}(y, x), \texttt{uscity}(x)\, \texttt{uscity}(y)$$

Nótese que hasta ahora se ha presentado el problema usando consultas en SQL. Sin embargo, cualquier consulta en SQL puede ser representada como consultas conjuntivas.

3.2. Contención de consultas conjuntivas

Un problema importante, relacionado a las consultas conjuntivas, consiste en determinar si una consulta está contenida o es equivalente a otra. Este concepto está estrechamente relacionado con el problema de encontrar una reescritura para una consulta usando vistas.

Definición 3.2 (Contención de consultas conjuntivas). Si Q_1 y Q_2 son consultas conjuntivas, se dice que $Q_1 \subseteq Q_2$ si para cualquier base de datos D, $Q_1(D)$ es un subconjunto de $Q_2(D)$.

Ejemplo 3.2. Considere las siguientes consultas conjuntivas sobre el esquema del sistema de vuelos del ejemplo 2.1:

$$\begin{aligned} Q_1(x1, z1) \ &: - \ \texttt{flight}(x1, y1), \texttt{flight}(y1, z1) \\ Q_2(x2, z2) \ &: - \ \texttt{flight}(x2, y2), \texttt{flight}(w2, z2) \end{aligned}$$

Q_1 produce vuelos desde la ciudad $x1$ hasta la ciudad $z1$, que pasan por una ciudad intermedia. Q_2 produce pares de ciudades (a, b) tales que a es salida de un vuela y b es llegada de un vuelo.

Puesto que siempre que exista un vuelo de x a y debe entonces existe un vuelo a partir de x y que llega a y entonces toda tupla producida por Q_1 también es producida por Q_2. Es decir, $Q_1 \subseteq Q_2$.

El problema de decidir si una consulta está contenida en otra es NP-completo, si las consultas no tienen negación ni predicados aritméticos [CM77]. Para probar la contención de dos consultas conjuntivas, se han propuesto varios métodos [Ull00, LMS95]. De éstos métodos, el de *"containment mappings"* se relaciona con este trabajo.

Containment mappings

La definición 3.3 presenta la definición de *containment mappings*. Posteriormente, el teorema 3.1 establece la conexión que existe entre *containment mapping* y la contención de consultas conjuntivas. En [LMS95] se muestra que los *Containment mapping* permiten determinar los posibles usos de una vista para responder una consulta.

Definición 3.3 (*Containment mappings* [CM77, LMS95]). Un *containment mapping* de las variables de Q_1 a Q_2 es una función de $Vars(Q_1) \rightarrow Vars(Q_2)$ tal que:

- Todo átomo de Q_1 se corresponde en algún átomo de Q_2.
- La cabeza de Q_1 se corresponde en la cabeza de Q_2.

Teorema 3.1. *$Q_2 \subseteq Q_1$ si y solo si existe un mapping de Q_1 a Q_2.*

Bases de datos canónicas

Se presentan en [Ull00] y también proveen un método para chequear la contención de dos consultas conjuntivas, a través del teorema 3.2.

Definición 3.4 (Base de datos canónicas [Ull00]). La base de datos canónica para una consulta conjuntiva Q está formada por los hechos que se obtienen al transformar los átomos en el cuerpo de Q, cambiando las variables en constantes.

Teorema 3.2. *$Q_1 \subseteq Q_2$ si y solo si al aplicar Q_2 a la base de datos canónica de Q_1 es posible derivar la cabeza de Q_1, cambiando las variables por constantes.*

3.3. El problema de responder consultas usando vistas

A continuación se introduce el problema de responder consultas usando vistas [GMPQ$^+$97, CGLV03, CGLV00, Hal00, LRO96b, Len02, MLF00, PH01]. Dada una consulta conjuntiva $Q(\vec{X}) : -r_0(\vec{X}_0), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)$ y un conjunto de vistas $\mathcal{V} = V_0, V_1, \ldots, V_n$ donde cada $V_i(\vec{Y}_i) = p_{0_i}(\vec{Y}_{0_i}), p_{1_i}(\vec{Y}_{1_i}), \ldots, p_{m_i}(\vec{Y}_{m_i})$, se desea encontrar una reescritura de Q usando las vistas $\mathcal{V}$.

Definición 3.5 (Reescritura de una consulta usando vistas [GMPQ$^+$97, Hal00, LRO96b, Len02, LRO96b, MLF00, PH01]). Una consulta conjuntiva Q' es una reescritura de Q, que usa las vistas $\mathcal{V} = V_1, V_2, \cdots, V_m$ si

- La expansión E de Q', está contenida en Q.
- Q' sólo contiene ocurrencias de $\mathcal{V}$.

Donde la expansión de una reescritura Q' se obtiene al reemplazar los átomos $V_i(\vec{Y}_i)$, en el cuerpo de Q', por la definición de la vista correspondiente.

Ejemplo 3.3. Considere el esquema del sistema de vuelos y la consulta del ejemplo 3.1.

$$Q(x, y) \text{ :- } \texttt{flight}(x, y), \texttt{flight}(y, x), \texttt{uscity}(x)$$

Donde la información sobre los vuelos se accede a través de las vistas presentadas en el ejemplo 2.1. Las mismas se pueden expresar, en notación de consultas conjuntivas de la siguiente manera:

$$\begin{aligned} \texttt{national}(x_1, y_1) &\text{ :- } \texttt{flight}(x_1, y_1), \texttt{uscity}(x_1), \texttt{uscity}(y_1) \\ \texttt{oneway}(x_2, y_2) &\text{ :- } \texttt{flight}(x_2, y_2) \\ \texttt{onestop}(x_3, z_3) &\text{ :- } \texttt{flight}(x_3, y_3), \texttt{flight}(y_3, z_3) \end{aligned}$$

Las siguiente es una posible reescrituras para Q:

$$R(x, y) \text{ :- } \texttt{oneway}(x, y), \texttt{national}(y, x)$$

ya que su expansión está contenida en Q.

$$E(x, z) \text{ :- } \texttt{flight}(x, y), \texttt{flight}(y, x), \texttt{uscity}(x), \texttt{uscity}(y)$$

Las definiciones que se presentan a continuación, dan la noción de cobertura de variables y subobjetivos de consultas conjuntivas. Estas definiciones se usarán posteriormente para la descripción de los algoritmos para resolver el problema de reescrituras.

Definición 3.6 (Cobertura de variables [Mit99]). Una vista $V(\vec{Y})$ cubre la variable x de la consulta Q si y solo si existe un *mapping* φ de Q a V tal que para todo i, j si $x = x_i = x_j$ entonces $\varphi(x_i) = \varphi(x_j)$.

Definición 3.7 (Cobertura de subobjetivos [Mit99]). Una vista $V(\vec{Y})$ cubre un subobjetivo $r_i(\vec{X}_i)$ de la consulta Q, usando el subobjetivo $p_k(\vec{Y}_k)$, mediante el *mapping* φ, si $p_k(\vec{Y}_k)$ está en el cuerpo de V y $p_k(\vec{Y}_k) = r_i(\varphi(\vec{X}_i))$.

Ejemplo 3.4. Para el siguiente problema de reescrituras sobre el ejemplo de vuelos:

$$\begin{aligned} Q(x, z) &:- \texttt{flight}(x, y), \texttt{flight}(y, z), \\ \texttt{oneway}(x_2, y_2) &:- \texttt{flight}(x_2, y_2) \end{aligned}$$

Es posible construir el siguiente *mapping* de variables entre la consulta y la vista `oneway`:

$$\varphi = \{x \to x_2, y \to y_2\}$$

Al construir φ es posible decir que la vista `oneway` cubre la variable x usando la variable x_2 y cubre al subobjetivo $\texttt{flight}(x, y)$ usando al subobjetivo $\texttt{flight}(x_2, y_2)$.

3.4. Tipos de reescrituras

Se distinguen dos tipos de reescrituras:

- Las reescrituras equivalentes, son aquellas cuya expansión es equivalente a la consulta original. Este tipo de reescritura es importante en optimización de consultas, pero no serán abordadas en este trabajo.
- Las reescrituras maximalmente contenidas, son útiles en sistemas de integración. Debido a que la información de las vistas puede ser incompleta, este tipo de reescrituras produce todas las respuestas posibles de una consulta dado un conjunto de vistas. A continuación se definen formalmente.

Definición 3.8 (Reescritura maximalmente contenida [LRO96b, PH01]). La reescritura Q', con respecto a $\mathcal{V} = V_1, V_2, \cdots, V_n$, es maximalmente contenida en Q, si para cualquier base de datos D, y sus extensiones $v = v_1, v_2, \cdots v_n$:

- $Q'(v) \subseteq Q(D)$.
- No existe otra consulta Q'' tal que $Q'(v) \subseteq Q''(v) \subseteq Q(D)$.

Dado una consulta conjuntiva Q y un conjunto de vistas conjuntivas $\mathcal{V}$, una reescritura maximalmente contenida puede llegar a ser una unión de reescrituras conjuntivas [PH01]. En el ejemplo 3.5 se presenta la reescritura maximalmente contenida del problema presentado en el ejemplo 3.3.

Ejemplo 3.5. Para el problema de reescrituras:

$$\begin{aligned} Q(x,y) &:- \texttt{flight}(x,y), \texttt{flight}(y,x), \texttt{uscity}(x) \\ \texttt{national}(x_1,y_1) &:- \texttt{flight}(x_1,y_1), \texttt{uscity}(x_1), \texttt{uscity}(y_1) \\ \texttt{oneway}(x_2,y_2) &:- \texttt{flight}(x_2,y_2) \\ \texttt{onestop}(x_3,z_3) &:- \texttt{flight}(x_3,y_3), \texttt{flight}(y_3,z_3) \end{aligned}$$

la reescritura maximalmente contenida para Q con respecto a las vistas $\{\texttt{national}, \texttt{oneway}, \texttt{onestop}\}$ es la unión de las siguientes reescrituras:

$$\begin{aligned} R_1(x,y) &:- \texttt{oneway}(x,y), \texttt{oneway}(y,x), \texttt{national}(x,w) \\ R_2(x,y) &:- \texttt{oneway}(x,y), \texttt{oneway}(y,x), \texttt{national}(w,x) \\ R_3(x,y) &:- \texttt{national}(x,y), \texttt{national}(y,x) \\ R_4(x,y) &:- \texttt{oneway}(x,y), \texttt{national}(y,x) \\ R_5(x,y) &:- \texttt{national}(x,y), \texttt{oneway}(y,x) \end{aligned}$$

3.5. Reescrituras minimales

Puede parecer por el ejemplo 3.5, que para encontrar las reescrituras a una consulta Q es necesario enumerar todas las soluciones potenciales y probar cuales de ellas se encuentran contenidas

en la consulta Q. Sin embargo, hay teoremas que limitan la búsqueda y muestran que el problema de reescribir una consulta en términos de las vistas es NP-completo [Ull00].

Definición 3.9 (Reescritura minimal [LMS95]). Una reescritura Q' para la consulta Q es minimal si:

- $Q' \subseteq Q$
- No existe otra reescritura Q'' para Q tal que:
  1. $Q' \subseteq Q'' \subseteq Q$.
  2. Q'' tiene menos subobjetivos que Q'.

La idea es que cualquier vista usada en una reescritura debe tener algún uso dentro de la consulta. Si alguna vista no cumple alguna función dentro de la consulta, debe ser eliminada. Además, todo subobjetivo de la consulta debe ser cubierto por alguna vista dentro de la reescritura. El teorema 3.3 muestra que toda reescritura para la consulta Q tiene a lo sumo el mismo número de subobjetivos que Q.

Teorema 3.3 ([LMS95]). *Si la consulta y las vistas son conjuntivos sin negación, ni comparaciones aritméticas o constantes en el cuerpo, entonces cualquier reescritura minimal tiene a lo sumo el mismo número de subobjetivos que Q.*

Teorema 3.4 ([LMS95]). *Decidir si existe una reescritura Q' de Q usando $\mathcal{V}$ tal que Q' contiene menos de k literales, donde k es el número de literales en Q, es NP-completo.*

Los teoremas 3.3 y 3.4 sugieren un algoritmo polinomial no determinístico para encontrar la reescritura maximalmente contenida de Q. Esto es:

- Buscar todas las reescrituras de tamaño menor a la cota.
- Tomar sólo aquellas que estén contenidas en Q.

El primer paso es complejo debido a que puede existir un número exponencial de posibles formas de cubrir los subobjetivos de la consulta usando los subobjetivos de las vistas.

Sin embargo, el teorema 3.5 muestra que el problema de reescrituras sigue siendo NP-completo aunque sólo exista una manera de cubrir la consulta usando las vistas.

Teorema 3.5 ([LMS95]). *Decidir si existe una reescritura Q' de Q usando $\mathcal{V}$ tal que Q' contiene menos de k literales, donde k es el número de literales en Q, aún cuando la consulta y las vistas no contienen predicados repetidos en el cuerpo, es NP-completo.*

Capítulo 4

Trabajo relacionado

El problema de reescritura de consultas ha sido abordado en varios contextos, en los cuáles se ha desarrollado mucho trabajo para el diseño de algoritmos eficientes para resolver el problema.

La nueva solución que se propone en el presente trabajo, consiste en transformar el problema en una teoría lógica en la que los modelos, estén en correspondencia con las reescrituras del mismo.

En este nuevo enfoque, se hizo uso de técnicas del área de compilación de conocimiento para transformar teorías en lógica proposicional en lenguajes de compilación adecuados que permitan hacer ciertas operaciones tratables. El lenguaje *Deterministic Decomposable Negation Normal Form* (d-DNNF) fue escogido para este trabajo, debido a que permite el conteo y enumeración de modelos de una teoría lógica en tiempo polinomial en el tamaño de la teoría compilada.

En la sección 4.1, se presentan los algoritmos desarrollados previamente para resolver el problema de reescritura. Luego, en la sección 4.2, se presenta una breve introducción al área de compilación de conocimiento y al lenguaje de compilación d-DNNF.

4.1. Algoritmos para reescritura de consultas

Como se presentó en el capítulo 3, existe una cota en el tamaño de las reescrituras para una consulta dada y por lo tanto existe una cota en el espacio de búsqueda de las reescrituras. Esta cota ha permitido el desarrollo de diferentes algoritmos que intentan explorar de manera eficiente el espacio de búsqueda [Hal00].

Varios algoritmos se han enfocado en el problema de responder consultas a través de vistas en el contexto de sistemas de integración de datos. Entre ellos se encuentran:

- *Inverse Rules* [Qia96, DG97a].
- *Bucket* [LRO96a, LRU96].
- *Query rewriting* [Mit99].
- *Minicon* [PH01].

Los últimos tres algoritmos trabajan en dos etapas. En la primera etapa, se identifican las vistas que pueden cubrir los subobjetivos de la consulta. Y, en una segunda etapa, se combinan estas vistas para formar las reescrituras del problema. El objetivo es reducir, en la primera etapa, la cantidad de vistas a considerar y de esta manera reducir el espacio de soluciones que el algoritmo debe evaluar.

En esta sección, se presentan brevemente los algoritmos, en particular, se presta especial atención al algoritmo de *Minicon*, sobre el cual se basó este trabajo.

4.1.1. *Inverse rules algorithm*

El *Inverse rules algorithm* consiste en la construcción de un conjunto de reglas que invierten las definiciones de las vistas, es decir, reglas que muestran como es posible calcular las tuplas de la base de datos a partir de las tuplas de las vistas. Para ver esto considere la vista `onestop`:

$$\texttt{onestop}(x_3, z_3) \quad :- \ \texttt{flight}(x_3, y_3), \texttt{flight}(y_3, z_3)$$

Para cada subobjetivo de la vista, se construye una regla invertida, de la siguiente manera:

$$\begin{aligned} \texttt{flight}(x_3, f(x_3, X)) \quad &: - \quad \texttt{onestop}(x_3, X) \\ \texttt{flight}(f(Z, z_3), z_3) \quad &: - \quad \texttt{onestop}(Z, z_3) \end{aligned}$$

El significado de estas reglas es que toda tupla de la forma (x_3, z_3) en la vista `onestop` es testigo de tuplas en la relación `flight` de la base de datos. Es decir, las tuplas en `flight` se forman de la siguiente manera:

- `flight` contiene una tupla (x_3, W) para algún valor de W.
- `flight` contiene una tupla (W, z_3) para algún valor de W.

Para expresar que el valor de W debe ser el mismo en ambos subobjetivos, se usa el término funcional $f(x_3, z_3)$. En general, se crea un símbolo funcional por cada variable existencial que aparezca en las vistas. En este caso, aunque existan varios valores de W en `onestop` que hagan que se cumpla el *join* en la vista, solo es necesario que exista uno de estos valores.

Finalmente, la reescritura de una consulta Q usando el conjunto de vistas $\mathcal{V}$ es el programa datalog que incluye la consulta Q y las reglas invertidas para cada una de las vistas en $V \in \mathcal{V}$.

Mas aún, el algoritmo de reglas invertidas es capaz de calcular la reescritura maximalmente contenida de Q usando $\mathcal{V}$ [DG97a]. El ejemplo 4.1 ilustra la manera como este algoritmo calcula la reescritura de una consulta usando las vistas.

Ejemplo 4.1. Supóngase que se tiene el siguiente problema de reescrituras, donde Q busca vuelos que pasan por tres ciudades intermedias:

$$\begin{aligned} Q(v, z) \ &:- \ \texttt{flight}(v, w), \texttt{flight}(w, x), \texttt{flight}(x, y), \texttt{flight}(y, z) \\ \texttt{onestop}(x_3, z_3) &:- \ \texttt{flight}(x_3, y_3), \ \texttt{flight}(y_3, z_3) \end{aligned}$$

y la vista `onestop` contiene las siguientes tuplas:

$$\begin{aligned} E(\texttt{oneway}) = \ & \{(BOS, BWI), \\ & (NYC, BWI), \\ & (BWI, CCS)\} \end{aligned}$$

Al calcular las reglas invertidas para `onestop`, se obtienen las siguientes tuplas:

$$\begin{aligned} \mathcal{E} = \ \{ \ & (BOS, f(BOS, BWI)), \\ & (f(BOS, BWI), BWI), \\ & (NYC, f(NYC, BWI)), \\ & (f(NYC, BWI), BWI), \\ & (BWI, f(BWI, CCS)), \\ & (f(BWI, CCS), CCS)\} \end{aligned}$$

Al aplicar Q sobre $\mathcal{E}$, se obtiene como resultado las tuplas: $\{(BOS, CCS), (NYC, CCS)\}$.

4.1.2. *Bucket algorithm*

La idea básica de el algoritmo de *Bucket* es que el número de reescrituras a ser consideradas puede reducirse drásticamente si se considera cada subobjetivo de la consulta por separado, determinando qué vistas pueden ser relevantes para cada uno. De esta manera se reduce el espacio de búsqueda, ya que solo se considerarían reescrituras en las cuales las vistas son útiles para cubrir algún subobjetivo de la consulta.

El algoritmo consta de dos etapas, que se presentan a continuación.

Creación de *buckets*

En esta primera etapa se crea un *bucket* para cada subobjetivo de la consulta. El *bucket* para un subobjetivo g contiene las vistas $V \in \mathcal{V}$ que contienen algún subobjetivo g' que pueda cubrir g. Además, se debe cumplir que toda variable distinguible en la consulta sea cubierta por una variable distinguible en la vista, antes de agregarla a un *bucket*.

Nótese que un subobjetivo g de la consulta puede cubrirse con más de un subobjetivo de una misma vista V, por lo cual, el *bucket* de g puede contener más de una ocurrencia de V.

En el ejemplo 4.2, se presentan los *buckets* que encuentra el algoritmo de *Bucket* para el problema de reescrituras en el sistema de vuelos.

Ejemplo 4.2. Los *buckets* que encuentra el algoritmo de *Buckets* para el siguiente problema:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ \texttt{oneway}(x_2,y_2) &:- \texttt{flight}(x_2,y_2) \\ \texttt{onestop}(x_3,z_3) &:- \texttt{flight}(x_3,y_3), \texttt{flight}(y_3,z_3) \end{aligned}$$

son los siguientes:

Subobjetivo	$0 : \texttt{flight}(x,y)$	$1 : \texttt{flight}(y,z)$
Bucket	$\texttt{oneway}(x,y)$ $\texttt{onestop}(x,w)$	$\texttt{oneway}(y,z)$ $\texttt{onestop}(w,z)$

En cada *bucket* se encuentra solo la cabeza de cada vista, con las variables sustituidas apropiadamente para hacer la unificación del subobjetivo. Las otras variables se sustituyen por variables

libres.

Obsérvese que el *bucket* $\texttt{flight}(x, y)$ contiene a la vista `oneway` dado que el *mapping* $\{x \to x_2, y \to y_2\}$ unifica este subobjetivo de la consulta con el primer subobjetivo de la vista.

Combinación de *buckets*

En esta etapa, el algoritmo forma las reescrituras combinando un subobjetivo de cada *bucket* creado en la fase anterior. Para cada una de estas reescrituras se chequea si se encuentra contenida en la respuesta. Si es así, la reescritura se añade a la respuesta.

La principal ineficiencia del algoritmo es que al considerar cada subobjetivo de la consulta por separado es posible agregar vistas que son irrelevantes en los *buckets*. Además, como ya se mostró en el capítulo 3, chequear si una consulta está contenida dentro de otra es un problema $NP - completo$. Por lo tanto, en la fase de combinación, el algoritmo puede llegar a ser ineficiente.

Ejemplo 4.3. Las reescrituras que encuentra el algoritmo de *Buckets* para el siguiente problema:

$$\begin{aligned} Q(x, z) &:- \texttt{flight}(x, y), \texttt{flight}(y, z), \\ \texttt{oneway}(x_2, y_2) &:- \texttt{flight}(x_2, y_2) \\ \texttt{onestop}(x_3, z_3) &:- \texttt{flight}(x_3, y_3), \texttt{flight}(y_3, z_3) \end{aligned}$$

son las siguientes:

$$\begin{aligned} Q_1(x, z) &:- \texttt{onestop}(x, w_1), \texttt{onestop}(w_2, z) \\ Q_2(x, z) &:- \texttt{onestop}(x, w_1), \texttt{onestop}(y, z) \\ Q_3(x, y) &:- \texttt{oneway}(x, y), \texttt{onestop}(w_2, z) \\ Q_4(x, y) &:- \texttt{oneway}(x, y), \texttt{oneway}(y, z) \end{aligned}$$

Debido a que, en la primera fase, se consideran los subobjetivos de la consulta por separado se consiguen reescrituras irrelevantes que hacen que la segunda fase sea más costosa.

Más aun, nótese que la reescritura:

$$Q_5(x, z) :- \texttt{onestop}(x, z)$$

no es producida por el algoritmo, debido a que no se está considerando la interacción entre los subobjetivos de la vista para cubrir la consulta.

4.1.3. *Query Rewriting Algorithm*

Similar al *Bucket algorithm*, el *Query Rewriting Algorithm* consta de tres fases:

- Construcción de *buckets*.
- Combinación de *buckets*.
- Minimización de soluciones.

Sin embargo, este algoritmo reduce el espacio de soluciones consideradas y evita el chequeo de contención de las reescrituras en la consulta, haciendo la construcción de los *buckets* de manera diferente. Además, este algoritmo define una fase posterior de minimización de soluciones, en el cual se eliminan subobjetivos superfluos de las reescrituras. A continuación se describen estas tres fases.

Creación de *buckets*

En esta fase se crean dos tipos de *buckets*:

- *Single Subgoal Buckets*: Representa las vistas que cubren un único subobjetivo.

  Una vista V puede insertarse en el *bucket* del subobjetivo $r_i(\vec{X}_i)$ si existe un subobjetivo $p_j(\vec{Y}_j)$ en V tal que:

  - Es posible cubrir $r_i(\vec{X}_i)$ usando $p_j(\vec{Y}_j)$.
  - Si una variable $X \in \vec{X}_i$ es cubierta usando una variable existencial en la vista V, entonces esta variable no puede ser una variable de *join* (compartida).

- *Shared Variable Buckets*: Representa las vistas que cubren los subobjetivos que comparten una variable en común.

  Una vista V puede insertarse en el *bucket* de la variable (compartida) X si para todo subobjetivo $r_i(\vec{X}_i)$ donde ocurra X se tiene que:

- Existe un subobjetivo $p_j(\vec{Y}_j)$ en V que cubre a $r_i(\vec{X}_i)$.
- Para toda variable $X' \in \vec{X}_i$ que sea cubierta usando una variable existencial en la vista V, se tiene que todo subobjetivo donde ocurra X' es cubierto en el *bucket*.

Ejemplo 4.4. Los *buckets* que encuentra el algoritmo de *Query Rewriting* para el siguiente problema de reescrituras:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ \texttt{oneway}(x_2,y_2) &:- \texttt{flight}(x_2,y_2) \\ \texttt{onestop}(x_3,z_3) &:- \texttt{flight}(x_3,y_3), \texttt{flight}(y_3,z_3) \end{aligned}$$

son los siguientes:

Subobjetivo	$\texttt{flight}(x,y)$	$\texttt{flight}(y,z)$	$y : \texttt{flight}(x,y), \texttt{flight}(y,z)$
Bucket	$\texttt{oneway}(x,y)$	$\texttt{oneway}(y,z)$	$\texttt{onestop}(x,z)$

La vista `onestop` no puede incluirse en el *bucket* del subobjetivo $\text{arc}(x,y)$ ya que la variable y_3 es existencial y estaría cubriendo a la variable y que forma parte de un *join* en la consulta.

El último *bucket* creado para la variable de *join* y aparecen en dos subobjetivos. En este *bucket* se incluye la vista `onestop`.

Combinación de *buckets*

En la segunda fase se combinan las vistas que se encuentren en un conjunto de *buckets* tal que cada subobjetivo de la consulta tenga un único *bucket* representándolo. Luego, una vista es seleccionada de cada *bucket* en el conjunto y la solución se genera creando una conjunción de estas vistas.

Ejemplo 4.5. Las reescrituras que encuentra el algoritmo de *Buckets* para siguiente problema:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ \texttt{oneway}(x_2,y_2) &:- \texttt{flight}(x_2,y_2) \\ \texttt{onestop}(x_3,z_3) &:- \texttt{flight}(x_3,y_3), \texttt{flight}(y_3,z_3) \end{aligned}$$

son las siguientes:

$$\begin{aligned} q_1(x,z) &:- \texttt{onestop}(x,z) \\ q_2(x,y) &:- \texttt{oneway}(x,y), \texttt{oneway}(y,z) \end{aligned}$$

Nótese para este ejemplo, que las reescrituras producidas son exactamente la reescritura maximalmente contenida descrita en el capítulo 3. Al agregar los *buckets* de variables compartidas, el algoritmo no produce las reescrituras superfluas que produce el algoritmo de *Buckets*. Aunque, en este algoritmo se producen más *buckets* y la fase de combinación es más costosa, se elimina el chequeo de contención de consultas conjuntivas que tiene el algoritmo de *Buckets*.

Minimización de soluciones

En la última fase, el algoritmo intenta minimizar las reescrituras que produce. Para ello identifica reescrituras en las que la misma vista aparece más de una vez. Luego, el algoritmo trata de colapsar las vistas en una sola ocurrencia, cuando sea posible.

4.1.4. *Minicon Algorithm*

Este algoritmo, también inspirado en el *Bucket algorithm*, trata de reducir el espacio de búsqueda de las reescrituras, encontrando para cada subobjetivo q_i de la consulta qué subobjetivos en la vista V pueden cubrir q_i. El algoritmo debe determinar el mínimo conjunto de subobjetivos de Q que debe cubrir la vista V_i para poder ser usada en una reescritura válida.

Consideremos la reescritura Q' de Q con respecto a $\mathcal{V}$. Dado que Q' es una reescritura para Q entonces existe un *containment mapping* θ_Q de Q a la expansión E' de Q', tal que $\theta_Q(Q(\vec{X})) = Q'(\vec{X'})$, es decir, toda variable distinguible en Q se corresponde en una variable distinguible en Q'.

Sea G_i el conjunto de subobjetivos de Q que se cubren con algún subobjetivo de la vista V_i en el *mapping* θ_Q. Sea θ_i igual al *mapping* θ_Q restringido a las variables de los subobjetivos de G_i.

Dado que θ_i es una restricción de θ_Q entonces G_i debe satisfacer que si $\theta_i(x)$ es una variable existencial en V entonces todo subobjetivo que incluya a x debe estar en G_i.

Por lo tanto, es posible desintegrar el *mapping* θ_Q en componentes disjuntas formadas por $(h_i, V_i, \varphi_i, G_i)$ que al ser combinadas pueden formar la reescritura Q'. Este es el objetivo del algoritmo de *Minicon* encontrar y combinar estas componentes, que sus autores llamaron *MCD*.

Un *MCD* (*Minicon description*) representa la manera como debe participar una vista dentro de una reescritura válida. Esto es, define la forma como deben cubrirse las variables de la consulta usando las variables de la vista y el subconjunto mínimo de subobjetivos de la consulta que debe cubrir la vista. A continuación se definen formalmente.

Definición 4.1 (*Minicon description MCD* [PH01]). Un *MCD* C para la consulta Q sobre la vista V es una tupla de la forma $(h_C, \varphi_C, \vec{Y}_C, V_C, G_C)$ donde:

- h_C es un homomorfismo de la cabeza de V.
- φ_C es un *mapping* parcial de las variables de Q en las variables de la vista V.
- G_C es un subconjunto de los subobjetivos de Q que son cubiertos por algún subobjetivo de $h_C(V)$.
- $\vec{Y}_C$ es el resultado de aplicar h_c a las variables distinguibles de V.

Para que un *MCD* sea válido para participar en una reescritura debe cumplir la siguiente propiedad.

Propiedad 1 ([PH01]). Sea C un *MCD* para Q sobre la vista V. Entonces C puede usarse en una reescritura no redundante para Q si se cumplen las siguientes condiciones:

C1.1 Para toda variable x distinguible en la consulta Q, que se encuentra en el dominio de φ_C, $\varphi_C(x)$ es distinguible en $h_C(V)$.

C1.2 Si $\varphi_C(x)$ es una variable existencial en $h_C(V)$, entonces para todo subobjetivo $g \in G_C$ que incluya a x se debe cumplir:

1. Todas las variables de g están en el dominio de φ_C.
2. $\varphi_C(g) \in h_C(V)$.

El algoritmo consta de dos etapas.

Creación de *MCDs*

La primera fase del algoritmo, calcula un conjunto $\mathcal{C}$ de *MCDs* minimales con respecto a la propiedad 1. La definición 4.2 muestra el significado de *MCD* minimal.

Definición 4.2 (*MCD* minimal [ABV06]). Un *MCD* válido $C = (V_C, h_C, \varphi_C, C)$ es minimal si no existe otro *MCD* válido $C' = (V'_C, h'_C, \varphi'_C, C')$tal que $C' \subset C$ y h'_C, φ'_C son restricciones de h_C, φ_C respectivamente.

El algoritmo 1 muestra la primera fase del algoritmo de *Minicon*. En esta primera etapa, se intenta cubrir los subobjetivos de la consulta usando subobjetivos de la vista. Una vez que encuentra un subobjetivo g de la consulta Q que puede ser cubierto por el subobjetivo v en la vista V, obtiene el *mapping* entre las variables de g_q y las de g_v. Luego, se construye un *MCD* para toda extensión de φ tal que G_C es un subconjunto minimal de subobjetivos de Q y G_C, φ_C y h_C cumplen la propiedad 1.

Algoritmo 1 [PH01] procedimiento **formMCDs**$(Q, \mathcal{V})$

Precondición: $\mathcal{Q}$ y $\mathcal{V}$ son consultas conjuntivas

1: $\mathcal{C} = \emptyset$
2: **for each** subobjetivo $g \in \mathcal{G}$ **do**
3: **for** $V \in \mathcal{V}$ y todo subobjetivo $v \in \mathcal{V}$ **do**
4: Sea h el homomorf. de cabeza sobre V tal que existe un mapping φ, tal que $\varphi(g) = h(v)$
5: **if** h y φ existen **then**
6: agregue a $\mathcal{C}$ todo MCD C que pueda se construido de la siguiente manera:
7: (a) φ_C, h_C son extensiones de φ, h respectivamente,
8: (b) G_C es el conjunto mínimo de subobjetivos de Q tal que G_C, φ_C y h_C satisfacen la Propiedad 1, y
9: (c) No es posible extender φ y h en φ'_C y h'_C tales que (b) se satisface y G'_C, definido como en (b), es un subconjunto de G_C
10: **end if**
11: **end for**
12: **end for**
13: **return** $\mathcal{C}$

En el ejemplo 4.6, se presentan los *MCDs* que encuentra el algoritmo de *Minicon* para el problema del ejemplo 3.3.

Ejemplo 4.6. Los *MCDs* que encuentra el algoritmo de *Minicon* para el siguiente problema de

reescrituras:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ \texttt{oneway}(x_2,y_2) &:- \texttt{flight}(x_2,y_2) \\ \texttt{onestop}(x_3,z_3) &:- \texttt{flight}(x_3,y_3), \texttt{flight}(y_3,z_3) \end{aligned}$$

son los siguientes:

MCD	*V*	φ	*h*	*G*
C_0	`oneway`	$\{x \to x_2, y \to y_2\}$	$\{\}$	$\{0\}$
C_1	`oneway`	$\{y \to x_2, z \to y_2\}$	$\{\}$	$\{1\}$
C_2	`onestop`	$\{x \to x_3, y \to y_3, z \to z_3\}$	$\{\}$	$\{0,1\}$

Combinación de *MCDs*

En esta fase, el algoritmo considera combinaciones de *MCDs* y para cada combinación válida crea una reescritura conjuntiva. Finalmente, la solución para el problema de reescritura es la unión de reescrituras conjuntivas.

La propiedad 2 afirma que el algoritmo de *Minicon* solo debe considerar combinaciones de *MCDs* que cubran todos los subobjetivos de la consulta exactamente una vez. No es necesario chequear si cada reescritura creada se encuentra contenida en Q. Por la forma en que se construyen los *MCDs* se garantiza que las reescrituras formadas por el algoritmo son reescrituras que van a estar contenidas en la consulta.

Propiedad 2 ([PH01]). Dada una consulta Q, un conjunto de vistas $\mathcal{V}$ y un conjunto de *MCDs* $\mathcal{C}$ para Q usando las vistas $\mathcal{V}$, las combinaciones de *MCDs* que producen reescrituras válidas, son de la forma $C_1, \ldots, C_l$, donde:

C2.1 $\bigcup\{C_i : 1 \leq i \leq l\} = Subob(Q)$

C2.2 $C_i \cap C_j = \emptyset$ para todo $1 \leq i < j \leq l$.

El algoritmo 2 muestra la fase de combinación del algoritmo de *Minicon*.

Algoritmo 2 [PH01] procedimiento **combineMCDs**($\mathcal{C}$)

Precondición: $\mathcal{C}$ conjunto de MCDs formados en la primera fase.

1: Dado un conjunto de MCDs $\mathcal{C}'$, considere la relación de equivalencia EC menos restrictiva consistente con todas las relaciones de equivalencias inducidas por φ_{C_i} para todo $C_i \in \mathcal{C}'$.
2: Sea $Resp = \emptyset$
3: **for each** $\mathcal{C}' = \{C_1, \ldots, C_n\} \subseteq \mathcal{C}$, $G_{C_1} \cup \cdots \cup G_{C_n} = Subob(Q)$ y si $i \neq j, G_{C_i} \cap G_{C_j} = \emptyset$ **do**
4: Defina Ψ_i de la siguiente manera:

$$\Psi_i(x) = \begin{cases} \varphi_i^{-1}(x) & \text{Si existe } x = \varphi_i(y) \\ \text{Nueva variable} & \text{Si no} \end{cases}$$

5: Cree la reescritura $Q'(EC(\vec{X})) : -V_{C_1}(EC(\Psi_1(\vec{Y}_{C_1}))), \ldots, V_{C_n}(EC(\Psi_n(\vec{Y}_{C_n})))$
6: Agregue Q' a $Resp$
7: **end for**
8: **return** $Resp$

Ejemplo 4.7. Las reescrituras que encuentra el algoritmo de *Minicon* para el siguiente problema de reescrituras en el sistema de vuelos:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ \texttt{oneway}(x_2,y_2) &:- \texttt{flight}(x_2,y_2) \\ \texttt{onestop}(x_3,z_3) &:- \texttt{flight}(x_3,y_3), \texttt{flight}(y_3,z_3) \end{aligned}$$

son las siguientes:

$$\begin{aligned} Q_1(x,z) &:- \texttt{onestop}(x,z) \\ Q_2(x,y) &:- \texttt{oneway}(x,y), \texttt{oneway}(y,z) \end{aligned}$$

Se debe enfatizar que aunque en [PH01] se asegura que *Minicon* produce la reescritura maximalmente contenida para Q, no hay garantía de que las reescrituras generadas sean minimales. Adicionalmente, la fase de combinación del algoritmo de *Minicon* puede generar reescrituras redundantes. De hecho, no es difícil diseñar problemas en los cuales *Minicon* genere reescrituras no minimales y/o redundantes. Para ello, se presenta el ejemplo 4.8.

Ejemplo 4.8. Las reescrituras que encuentra el algoritmo de *Minicon* para el siguiente problema de

reescrituras en el sistema de vuelos:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ V_1(x_1,y_1) &:- \texttt{flight}(x_1,x_1), \texttt{flight}(x_1,y_1)\,, \\ V_2(x_2,y_2) &:- \texttt{flight}(x_2,y_2)\,. \end{aligned}$$

son las siguientes:

$$\begin{aligned} R_1(x,z) &:- V_1(x,y),\ V_1(y,z)\,, \\ R_2(x,z) &:- V_2(x,y),\ V_2(y,z)\,, \\ R_3(x,z) &:- V_1(x,y),\ V_2(x,z)\,, \\ R_4(x,z) &:- V_1(x,y),\ V_2(y,z)\,, \\ R_5(x,z) &:- V_2(x,z),\ V_1(z,y)\,, \\ R_6(x,z) &:- V_2(x,y),\ V_1(y,z)\,, \\ R_7(x,z) &:- V_1(x,z),\ V_1(x,y)\,, \\ R_8(x,z) &:- V_1(x,z),\ V_1(z,y)\,, \\ R_9(x,x) &:- V_1(x,y),\ V_1(x,z) \end{aligned}$$

Sin embargo, obsérvese las reescrituras R_7, R_8, R_9. En primer lugar, es posible eliminar el segundo subobjetivo de R_7. Esto se debe a que las variables z y y cumplen la misma función en la reescritura, dado que y es libre. Por lo tanto, los subobjetivos $V_1(x,z)$ y $V_1(x,y)$ producen las mismas tuplas. Luego, R_7 debe ser:

$$R_7'(x,z) \ :- \ V_1(x,z)$$

Más aún las reescrituras R_8, R_9 se encuentran contenidas dentro de R_7'. De lo cual se concluye

que, en realidad, la reescritura maximalmente contenida debe ser la siguiente:

$$
\begin{aligned}
R_1(x,z) &:- V_1(x,y),\ V_1(y,z)\,,\\
R_2(x,z) &:- V_2(x,y),\ V_2(y,z)\,,\\
R_3(x,z) &:- V_1(x,y),\ V_2(x,z)\,,\\
R_4(x,z) &:- V_1(x,y),\ V_2(y,z)\,,\\
R_5(x,z) &:- V_2(x,z),\ V_1(z,y)\,,\\
R_6(x,z) &:- V_2(x,y),\ V_1(y,z)\,,\\
R'_7(x,z) &:- V_1(x,z)
\end{aligned}
$$

Implementación del algoritmo

Hasta ahora, en esta sección, se ha descrito el algoritmo de *Minicon* en base a sus definiciones. Sin embargo, es importante resaltar que en [PH01] no se describe la manera exacta como se debe implementar la primera fase del algoritmo.

Recuérdese que para generar los *MCDs*, es necesario conseguir un *mapping* φ_C entre un subobjetivo g_q de la consulta y un subobjetivo g_v de la vista V. Luego, este *mapping* se debe extender de manera de que se pueda cubrir un conjunto minimal de subobjetivos G_C en Q.

El problema es que en [PH01] no se describe la manera como se va a extender φ_C ni como se va a decidir cual es el conjunto minimal de subobjetivos que se pueden cubrir de manera que se cumpla la propiedad 1.

Hasta el momento, se observaron dos maneras de hacer la implementación del algoritmo.

- En la implementación presentada en [ALU01] se generan todos los posibles subconjuntos de los subobjetivos de la consulta y la vista.

  Para cada par de subconjuntos generados se intenta conseguir un *mapping* φ que los unifique y que cumpla la propiedad 1. Si el *mapping* existe, se crea un *MCD* C_1. Luego, C_1 se agrega a la lista si no existe otro *MCD* C_2 tal que $G_{C_2} \subseteq G_{C_1}$.

- En la implementación del algoritmo realizada en este trabajo, se genera un *mapping* inicial φ.

1. Si con este *mapping* se puede cubrir un conjunto de subobjetivos G que cumpla la propiedad 1, se generan subconjuntos minimales de G y para cada uno de ellos se construye un *MCD*.
2. Si no, se extiende φ hasta conseguir un conjunto de subobjetivos que cumplan la propiedad 1.
3. Debido a que se expanden los *mappings* agregando variables, no se puede garantizar que todos los *MCDs* generados sean minimales con respecto a otros *mappings*. Por lo cual, luego de generar los *MCDs* se procede a eliminar los que no sean minimales. Esta eliminación no es compleja, tiene orden $O(n^2)$ siendo n el número de *MCDs* generados.

Estas dos implementaciones tienen comportamientos diferentes en diferentes circunstancias. La implementación de [ALU01], al generar todos los subconjuntos de la consulta y las vistas, se comporta peor cuando crece el número de subobjetivos, de la consulta y/o las vistas. Por otro lado, el comportamiento de la implementación de *Minicon* realizada en este trabajo, tiene problemas cuando hay muchas variables en los cuerpos de las consultas y las vistas, ya que va generando todos los *mappings* de variables posibles.

4.2. Compilación de Conocimiento

Compilación de Conocimiento es un área reciente de inteligencia artificial que trata el problema de transformar teorías de la lógica proposicional en un lenguaje de compilación apropiado que permita realizar ciertas operaciones en tiempo polinomial en el tamaño de la teoría compilada [CD97].

En general, el proceso de compilación es intratable y su costo es amortizado si el número de operaciones sobre la teoría compilada es suficientemente grande. Sin embargo, aunque la compilación tiene orden exponencial en tiempo y espacio en el peor caso, no es necesariamente así en promedio. Por lo cual, ya se han usado teorías compiladas en verificación formal [EMCGP99] y planificación [GT99, PBDG05].

4.2.1. *Decomposable Negation Normal Form*

DNNF es un lenguaje de compilación reciente creado por Darwiche en el año 2001 [Dar01]. Soporta un gran número de operaciones en tiempo lineal, entre ellas: enumeración de modelos.

Como se verá mas adelante, la enumeración de modelos de una teoría en lógica proposicional es una operación necesaria para este trabajo.

Definición 4.3 (*Negation Normal Form* NNF [Dar01]). Una fórmula proposicional está en NNF si está formada sólo por conjunciones y disyunciones y las negaciones están junto a los literales.

Una fórmula en NNF puede representarse como un grafo dirigido acíclico con raíz, como el que se presenta en la figura 4.1, en el cual las hojas se etiquetan con literales, *true* ó *false* y los nodos internos se etiquetan con $\wedge$ ó $\vee$.

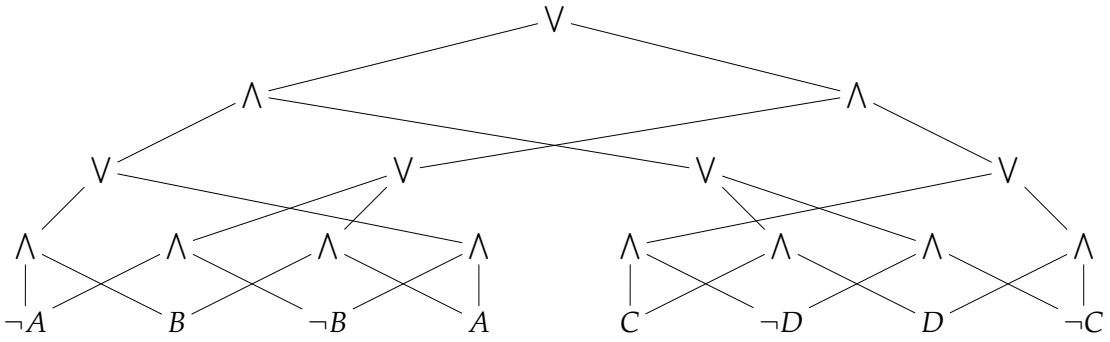

Figura 4.1: Una fórmula en *Negation Normal Form* NNF representada como un grafo dirigido acíclico con raíz.

Definición 4.4 (*Decomposable Negation Normal Form* DNNF [Dar01]). Un DNNF es una fórmula en NNF que además satisface la siguiente propiedad: Para toda conjunción C en el NNF, cada término de C no comparte ninguna variable.

El NNF de la figura 4.1 es también un DNNF, ya que ninguna de sus 10 conjunciones comparte alguna variable. Esta propiedad es la que hace al DNNF tratable, ya que permite descomponer ciertas operaciones con respecto al NNF en operaciones más pequeñas con respecto a sus subsentencias [Dar98]. Es posible determinar si un DNNF es satisfacible con un recorrido de abajo hacia arriba sobre su DAG.

Definición 4.5 (*Deterministic Decomposable Negation Normal Form* d-DNNF [Dar01]). Un d-DNNF es una fórmula en DNNF que además satisface la siguiente propiedad: Para toda disyunción C en el NNF, los términos de C son lógicamente contradictorios.

El NNF de la figura 4.1 es también un d-DNNF. Por ejemplo, el nodo OR más a la izquierda tiene dos hijos: las sub-fórmulas $\neg A \wedge B$ y $A \wedge \neg B$. La conjunción de estas fórmulas es lógicamente contradictoria y los mismo sucede para los demás nodos OR.

La forma normal d-DNNF es más tratable que DNNF ya que además permite contar modelos, dada una instanciación cualquiera de variables, en tiempo polinomial. Sin embargo, garantizar *Decomposability* es difícil y, más aún, garantizar *Determinism* sin perder *Decomposability* [Dar98].

Definición 4.6 (*Smooth Deterministic Decomposable Negation Normal Form* d-DNNF [Dar01]). Un sd-DNNF es una fórmula en d-DNNF que además satisface la siguiente propiedad: Para toda disyunción C en el NNF, cada término de C menciona las mismas variables.

Smoothness es fácil de garantizar y además facilita algunas operaciones sobre el DNNF.

4.2.2. Compilación de CNF a DNNF

Antes de describir el algoritmo más sencillo de transformación de CNF a DNNF, es necesario dar la definición de condicionamiento de una teoría proposicional.

Definición 4.7 (Condicionamiento [Dar01]). Sea Δ una teoría en lógica proposicional y α una instanciación de variables. El condicionamiento de Δ dado α es una teoría que se obtiene al reemplazar cada variable de Δ por *True*/*False* si es consistente/inconsistente con α.

Por ejemplo, condicionar el DNNF $\Delta_1 = (\neg A \wedge \neg B) \vee (B \wedge C)$ dado $\alpha_1 = \neg B \wedge D$ produce el DNNF $\Delta_2 = (\neg A \wedge \mathit{True}) \vee (\mathit{False} \wedge C)$.

A continuación se presenta el algoritmo recursivo `DNNF` que convierte cualquier CNF, en forma clausal, en un DNNF [Dar01].

1. Si Δ contiene una única cláusula α: $\texttt{DNNF}(\Delta) = \alpha$.
2. Si no, $\texttt{DNNF}(\Delta) = \bigvee_\beta \texttt{DNNF}(\Delta_1|\beta) \wedge \texttt{DNNF}(\Delta_2|\beta) \wedge \beta$

   donde Δ_1, Δ_2 es una partición de las cláusulas de Δ y β son todas las posibles asignaciones de las variables que comparten Δ_1 y Δ_2.

Por ejemplo, considere $\Delta = \{\neg A \vee B, \neg B \vee C\}$ y $\Delta_1 = \{\neg A \vee B\}$, $\Delta_2 = \{\neg B \vee C\}$. Entonces,

se tiene que:

$$\begin{aligned}
\texttt{DNNF}(\Delta) &= \bigvee_{\beta} \texttt{DNNF}(\Delta_1|\beta) \wedge \texttt{DNNF}(\Delta_2|\beta) \wedge \beta \\
&= (\texttt{DNNF}(\Delta_1|B) \wedge \texttt{DNNF}(\Delta_2|B) \wedge B) \vee (\texttt{DNNF}(\Delta_1|\neg B) \wedge \texttt{DNNF}(\Delta_2|\neg B) \wedge \neg B) \\
&= ((\neg A \vee \mathit{True}) \wedge (\mathit{False} \vee C) \wedge B) \vee ((\neg A \vee \mathit{False}) \wedge (\mathit{True} \vee C) \wedge \neg B) \\
&= (C \wedge B) \vee (\neg A \wedge \neg B)
\end{aligned}$$

El algoritmo DNNF convierte cualquier teoría en forma clausal en un DNNF equivalente pero a costa de aumentar el tamaño de la teoría. El modo en que se divida la teoría en Δ_1 y Δ_2 influye en el tamaño. Pero el algoritmo presentado DNNF, no especifica como debe la división de la teoría. Para ello, se requiere hacer uso de árboles de descomposición, para representar un particionamiento recursivo de las cláusulas de Δ.

Definición 4.8 (Árbol de descomposición [Dar01]). Un árbol de descomposición T para una teoría en forma clausal Δ es un árbol binario cuyas hojas son las cláusulas de Δ. Si t es una hoja de T que corresponde a la cláusula α en Δ, entonces $\Delta(t) = \alpha$. Para todo nodo interno, se define t_{der} y t_{izq} como los subárboles derecho e izquierdo de t.

El árbol de la figura 4.2 para una teoría con las siguientes cláusulas $\{\neg A \vee B, \neg C \vee \neg A, \neg B \vee C\}$.

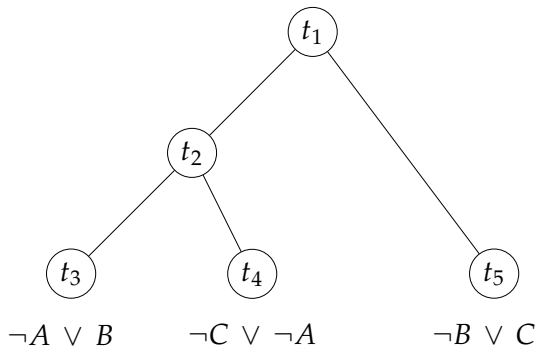

Figura 4.2: Un árbol de descomposición para la teoría $\{\neg A \vee B, \neg C \vee \neg A, \neg B \vee C\}$

Dado un árbol de descomposición para una teoría en forma clausal Δ, a continuación se presenta el algoritmo $\texttt{DNNF1}(t)$ para transformar Δ en un DNNF [Dar01].

1. Si t es una hoja: $\texttt{DNNF1}(t) = \Delta(t)$.
2. Si no, $\texttt{DNNF1}(t) = \bigvee_\beta \texttt{DNNF1}(t_{der}|\beta) \wedge \texttt{DNNF1}(t_{izq}|\beta) \wedge \beta$
   donde β son todas las posibles asignaciones de las variables que comparten t_{izq} y t_{der}.

El compilador `c2d`

Se han propuesto diferentes implementaciones para compilar CNF en DNNF [Dar00, Dar01, Dar02]. El compilador más reciente es `c2d` [Dar04].

Este último incorpora nuevas técnicas que mejoran significativamente el desempeño del compilador. Los dos avances más importantes presentes en `c2d` son los siguientes:

- *Backtracking* dirigido por conflicto a través de aprendizaje de cláusulas [SS96, MMZ$^+$01].
- Caracterización del estado de las cláusulas para reutilización (*caching*) de resultados intermedios.

4.2.3. Enumeración de modelos a partir de un DNNF

En la definición 4.9 se presenta la enumeración de modelos sobre un DNNF. Esta definición puede verse como un recorrido sobre el grafo que representa al DNNF.

Definición 4.9 (Enumeración de modelos de un DNNF [Dar01]). Dado un s-DNNF Δ, se define $\mathbf{Models}(\Delta)$ de la siguiente manera:

$$\mathbf{Models}(\Delta) = \begin{cases} \{\{p = True\}\} & \text{Si } \Delta = p \\ \{\{p = False\}\} & \text{Si } \Delta = \neg p \\ \{\{\}\} & \text{Si } \Delta = True \\ \{\} & \text{Si } \Delta = False \\ \cup_i \mathbf{Models}(\alpha_i) & \text{Si } \Delta = \bigvee_i \alpha_i \\ \{\cup_i \beta_i \ : \ \beta_i \in \mathbf{Models}(\alpha_i)\} & \text{Si } \Delta = \bigwedge_i \alpha_i \end{cases}$$

A partir de su definición, se observa que la complejidad de enumerar modelos a partir de un DNNF es $O(mn^2)$, en espacio y tiempo, donde m es el tamaño del DNNF, medido en cantidad de aristas en el DAG, y n es el número de modelos.

El problema de hacer una implementación basada en la definición anterior es que, si la teoría tiene un número considerable de modelos, la misma requeriría guardar en memoria todos los modelos antes de producir respuesta.

Una implementación alternativa consiste en obtener los modelos de manera iterativa, recorriendo el d-DNNF una vez por cada modelo de la teoría. Si se hace un recorrido DFS sobre el DAG del d-DNNF pero visitando solo un hijo de cada nodo OR se puede construir un modelo con las variables que encuentre en las hojas. De esta manera, la implementación iterativa tomaría espacio lineal en el tamaño del d-DNNF y tiempo lineal en el número de modelos.

La ventaja es que de esta manera se pueden ir generando y produciendo modelos, sin tener que guardarlos todos en memoria antes de poder dar la respuesta.

Capítulo 5

Solución propuesta

Este capítulo describe la nueva solución que se desarrolló en este trabajo, aplicando técnicas de representación del conocimiento al problema de reescritura de consultas usando vistas.

Se utilizó un enfoque basado en el algoritmo de *Minicon*, construyendo y combinando los *MCDs*, pero utilizando una perspectiva lógica. La idea es construir una teoría en lógica proposicional a partir del problema de reescrituras, que llamaremos teoría *MCD*, en la cual los modelos se encuentran en correspondencia con los *MCDs* que permiten construir las reescrituras.

Los *MCDs* pueden ser reconstruidos a partir de los modelos de la teoría y luego combinados como se describe en la segunda fase del algoritmo de *Minicon*. O es posible extender la teoría, como se describe en este capítulo, de manera que sus modelos estén en correspondencia con las reescrituras.

En la sección 5.1, se describe la construcción de la teoría lógica para creación de *MCDs* dada una consulta y un conjunto de vistas. Posteriormente, en la sección 5.2, se presenta la teoría lógica extendida para obtener las reescrituras del problema.

Por último, en la sección 5.3, se presenta la manera como se implementó la solución propuesta en este trabajo.

5.1. Teoría para la representación de *MCDs*

La fase inicial de creación de *MCDs* del algoritmo de Minicon es compleja debido a que puede existir un número exponencial de posibles *mappings* de la consulta a las vistas [LMS95]. Además, como se explicó en la sección 4.1.4, *Minicon* es difícil de implementar.

La idea de la creación de una teoría lógica para representación de *MCDs*, es poder encontrar los *MCDs* a partir de los modelos. Por esto se decidió que la primera aproximación de este trabajo consistiría en diseñar una teoría lógica en la cual un modelo de la teoría representara un *MCD* para una consulta Q sobre un conjunto de vistas $\mathcal{V}$.

Para el diseño de la teoría se consideró una simplificación para la representación de un *MCD*. En esta simplificación el *MCD* está formado por:

- La vista V usada en el *MCD*.
- El conjunto G de subobjetivos de Q que es cubierto por el *MCD*.
- Una relación τ : *Vars(Q)* $\times$ *Vars(V)*, en la cual un par (q, v) representa que la variable q de Q se cubre con la variable v de V. Se permite que existan variables de Q que correspondan a más de una variable en V, cumpliéndose la siguiente relación entre τ, φ y h:

  $$(q, v) \in \tau \Leftrightarrow \varphi(q) = h(v)$$

  Además, para cumplir con la propiedad de homomorfismo de cabeza de h, se impone la siguiente restricción: "Si en la relación τ la variable q de la consulta se cubre con más de una variable en V, entonces todas estas variables deben ser distinguibles en V".

Es importante notar que esta simplificación no pierde la información representada por el *MCD*. Obsérvese lo siguiente:

- El *mapping* h_C se puede obtener a partir de la relación τ. Siempre que en τ una variable q de la consulta se relacione un conjunto H_q de variables de V, entonces para todo $x \in H_q$, $h_C(x) = rep(H_q)$, donde $rep(H_q)$ es una variable cualquiera de H_q.

  Nótese que la restricción que se impone sobre τ garantiza que h_C cumple la propiedad de homomorfismos de la cabeza de la vista que sólo iguala variables distinguibles.

- El *mapping* φ_C se obtiene a partir de la relación τ. Para todas las variable q de la consulta que se correspondan con un conjunto H_q de variables de V, entonces $\varphi_C(q) = h_C(x)$, donde $x \in H_q$.

- $V(\vec{Y})_C$ se puede obtener aplicando el *mapping* h_C a V.

En base a esta simplificación, el apéndice A describe las variables y cláusulas que conforman la teoría lógica en la que un modelo representa un *MCD* para Q sobre un conjunto de vistas $\mathcal{V}$.

La teoría *MCD* T_M, para el problema de reescrituras de Q con respecto a $\mathcal{V}$, es la colección de cláusulas generadas a partir del problema, siguiendo las definiciones del apéndice A. La definición 5.1 describe la forma de obtener un *MCD* a partir de un modelo.

Definición 5.1 (*MCD* asociado a un modelo de la teoría MCD). Un modelo ω para la teoría T_M construida para $Q, \mathcal{V}$ representa:

- El *MCD* vacío, aquel que no cubre algún subobjetivo de Q, si se cumple que $v_{-1} = True$.
- El *MCD* C_ω construido a partir de ω de la siguiente manera:
  - **a.** $V_C = v_i$ para $v_i = True$.
  - **b.** $G_C = \{g_j : g_j = True\}$.
  - **c.** $\tau_C = \{(a, b) : t_{a,b} = True\}$.
  - **d.** h_C y ϕ_c se construyen a partir de τ_C.

El ejemplo 5.1 describe la teoría correspondiente a un problema de reescrituras sobre el sistema de vuelos. El teorema 5.1 indica que todo modelo de la teoría descrita en el ejemplo 5.1, es un *MCD* en el problema de reescrituras correspondiente.

Ejemplo 5.1. Para el siguiente problema de reescrituras:

$$\begin{aligned} Q(x, z) &:- \texttt{flight}(x, y), \texttt{flight}(y, z), \\ V_1(x_2, y_2) &:- \texttt{flight}(x_2, y_2) \end{aligned}$$

la teoría de *MCDs* correspondiente se presenta a continuación:

Variables

- v_i indica que el *MCD* usa la vista i. (Si el *MCD* es vacío, se usa la vista v_{-1}).

$$v_{-1} \quad v_1$$

- g_j indica que el *MCD* cubre al subobjetivo j de la consulta.

$$g_0 \quad g_1$$

- $z_{j,k,i}$ indica que el subobjetivo j de la consulta es cubierto por el subobjetivo k de la vista i.

  $$z_{0,0,1} \quad z_{1,0,1}$$

  Por ejemplo, $z_{0,0,1}$ significa que el subobjetivo $\texttt{flight}(x, y)$ de Q se cubre con el subobjetivo $\texttt{flight}(x_2, y_2)$ de V_1.

- $t_{a,b}$ indica que la variable a de Q se corresponde con la variable b de la vista.

  $$t_{0,0} \quad t_{1,0} \quad t_{2,0} \quad t_{1,1} \quad t_{2,1}$$

  Por ejemplo, $t_{0,0}$ significa que x se cubre con la variable x_2.

Cláusulas

- Por lo menos una vista debe usarse en el *MCD*.

  $$v_1$$

- Si un *MCD* usa la vista V_i entonces debe cubrir algún subobjetivo de la consulta.

  $$v_1 \Rightarrow (g_0 \vee g_1)$$

- Si un *MCD* usa la vista V_i y cubre el subobjetivo j de la consulta, entonces este subobjetivo de la consulta debe ser cubierto por algún subobjetivo de la vista V_i.

  $$v_1 \wedge g_0 \Leftrightarrow z_{0,0,1}$$
  $$v_1 \wedge g_1 \Leftrightarrow z_{1,0,1}$$

- Si se cubre el subobjetivo j de la consulta con el subobjetivo k de la vista V_i entonces todas las variables de $r_j(\vec{X}_j)$ se cubren con las variables $p_{k_i}(\vec{Y}_{k_i})$

  $$v_1 \wedge z_{0,0,1} \Rightarrow t_{0,0} \wedge t_{1,1}$$
  $$v_1 \wedge z_{1,0,1} \Rightarrow t_{1,0} \wedge t_{2,1}$$

- Si se tiene $\{x_a \rightarrow y_b\} \in \tau$, entonces debe existir un subobjetivo j en la consulta que pueda cubrirse con un subobjetivo k de la vista V_i tal que $x_l \in r_j(\vec{X}_j), y_b \in p_{k_i}(\vec{Y}_{k_i})$.

$$v_1 \wedge t_{0,0} \Rightarrow z_{0,0,1} \quad v_1 \wedge t_{1,0} \Rightarrow z_{1,0,1}$$
$$v_1 \wedge t_{1,1} \Rightarrow z_{0,0,1} \quad v_1 \wedge t_{1,2} \Rightarrow z_{1,0,1}$$

- Si se tiene $\{x_a \rightarrow y_b\} \in \tau$, entonces debe usarse la vista V_i a la cual pertenece la variable y_b.

$$t_{0,0} \Rightarrow v_0 \quad t_{1,0} \Rightarrow v_0$$
$$t_{1,1} \Rightarrow v_0 \quad t_{1,2} \Rightarrow v_0$$

Teorema 5.1. *Un MCD para este problema:*

$$Q(x,z) \text{ :- } \texttt{flight}(x,y), \texttt{flight}(y,z),$$
$$V_1(x_2,y_2) \text{ :- } \texttt{flight}(x_2,y_2)$$

es equivalente a un modelo en esta teoría:

$$\begin{array}{lll}
v_{-1} \quad v_1 & v_1 \wedge t_{0,0} \Rightarrow z_{0,0,1} & t_{0,0} \Rightarrow v_0 \\
v_1 \Rightarrow (g_0 \vee g_1) & v_1 \wedge t_{1,1} \Rightarrow z_{0,0,1} & t_{1,1} \Rightarrow v_1 \\
z_{0,0,0} \Rightarrow \neg z_{0,1,0} & v_1 \wedge t_{1,0} \Rightarrow z_{1,0,1} & t_{1,0} \Rightarrow v_1 \\
z_{1,0,0} \Rightarrow \neg z_{1,1,0} & v_1 \wedge t_{1,2} \Rightarrow z_{1,0,1} & t_{1,2} \Rightarrow v_1 \\
v_1 \wedge g_0 \Leftrightarrow z_{0,0,1} & v_1 \wedge z_{0,0,1} \Rightarrow t_{0,0} \wedge t_{1,1} & v_1 \wedge g_0 \Rightarrow \neg g_1 \\
v_1 \wedge g_1 \Leftrightarrow z_{1,0,1} & v_1 \wedge z_{1,0,1} \Rightarrow t_{1,0} \wedge t_{2,1} &
\end{array}$$

En general, el teorema 5.1 es cierto para cualquier instancia del problema de reescritura de consultas. El teorema 5.2 generaliza el teorema 5.1. Su demostración se presenta en el apéndice B.

Teorema 5.2. *Sea C un MCD para el problema de reescribir la consulta Q usando las vistas $\mathcal{V}$. Sea T_M la teoría generada para el problema. Si C es minimal, entonces existe un modelo ω para T_M tal que $C = C_\omega$. Si ω es un modelo para T_M entonces ω es vacío o C_ω es un MCD.*

Del teorema 5.2, se deriva que no es posible garantizar que todos los *MCDs* generados son minimales. Esto se muestra en el ejemplo 5.2.

Ejemplo 5.2. Los *MCDs* que produce la teoría *MCD* para el siguiente problema de reescrituras:

$$Q(x,z) \text{ :- } \texttt{flight}(x,y), \texttt{flight}(y,z),$$
$$V_1(x_1,y_1,w_1) \text{ :- } \texttt{flight}(x_1,y_1), \texttt{flight}(y_1,z_1), \texttt{flight}(y_1,w_1)$$

son los siguientes:

MCD	V	φ	h	G
C_0	V_1	$\{x \to x_1, y \to y_1\}$	$\{\}$	$\{0\}$
C_1	V_1	$\{x \to y_1, y \to z_1\}$	$\{\}$	$\{0\}$
C_2	V_1	$\{y \to x_1, z \to y_1\}$	$\{\}$	$\{1\}$
C_3	V_1	$\{y \to y_1, z \to z_1\}$	$\{\}$	$\{1\}$
C_4	V_1	$\{x \to x_1, y \to y_1, z \to z_1\}$	$\{\}$	$\{0,1\}$
C_5	V_1	$\{x \to y_1, y \to z_1, z \to y_1\}$	$\{x_1 \to y_1\}$	$\{0,1\}$

Obsérvese que los *MCDs* C_4 y C_5 no son minimales, ya que los *MCDs* C_0, C_3 están contenidos dentro de C_4 y C_1, C_2 están contenidos dentro de C_5.

Las cláusulas **Cl 14**, presentadas en el apéndice A, aunque no son necesarias para la correctitud de la teoría, reducen la cantidad de *MCDs* generados en el caso de que las vistas tengan todas las variables distinguibles. Sin embargo, a pesar de que se puede llegar a generar un número exponencial de modelos en la teoría, se sigue obteniendo una mejora en el tiempo de ejecución de la fase de generación de *MCDs*. Ésto se muestra en el estudio experimental.

Además, como se comentó en la sección 4.1.4, dependiendo de como se implemente, la primera fase del algoritmo de *Minicon* también produce *MCDs* que no son minimales. Por lo cual es necesaria una fase de eliminación intermedia, aunque no se reporta en [PH01].

5.2. Teoría extendida

Como se mencionó al principio de este capítulo, los *MCDs* pueden ser reconstruidos a partir de los modelos de la teoría MCD y luego combinados como se describe en la segunda fase del algoritmo de *Minicon*. También es posible extender la teoría MCD, de manera que sus modelos estén en correspondencia con las reescrituras.

Para ver como se puede extender la teoría MCD, recordar el teorema 3.3, presentado en el capítulo 3. Este teorema muestra que toda reescritura para Q tiene a lo sumo m subobjetivos, donde m es la cantidad de subobjetivos que Q. Dado que todo *MCD* no vacío debe cubrir por lo menos un subobjetivo, entonces, una reescritura puede ser formada con a lo sumo m *MCDs*.

En base a esto, la teoría extendida consiste en m copias de la teoría de *MCDs*, donde las variables de cada teoría son marcadas con el superíndice t que indica la copia a la que pertenece.

Además, se agregan cláusulas que garanticen que se cubren subobjetivos disjuntos de la consulta y que no se consideren reescrituras simétricas. Esto último se garantiza imponiendo un orden en las copias de las teorías de *MCDs*. Las cláusulas de la teoría extendida se definen en el apéndice C.

Ejemplo 5.3. Para el siguiente problema de reescrituras:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ V_1(x_2,y_2) &:- \texttt{flight}(x_2,y_2) \end{aligned}$$

la teoría lógica que representaría el problema de reescrituras está formada por 2 copias de la teoría de *MCDs*, que se presentó en el ejemplo 5.1.

Las variables son marcadas con el número de la copia de la teoría, de la siguiente manera:

$$\begin{array}{cccccccccc} v_{-1}^0 & v_1^0 & g_0^0 & g_1^0 & z_{0,0,1}^0 & z_{1,0,1}^0 & t_{0,0}^0 & t_{1,0}^0 & t_{2,0}^0 & t_{1,1}^0 & t_{2,1}^0 \\ v_{-1}^1 & v_1^1 & g_0^1 & g_1^1 & z_{0,0,1}^1 & z_{1,0,1}^1 & t_{0,0}^1 & t_{1,0}^1 & t_{2,0}^1 & t_{1,1}^1 & t_{2,1}^1 \end{array}$$

Las cláusulas se forman por dos copias de las cláusulas de la teoría de *MCDs*:

$$
\begin{array}{ccc}
v^0_{-1} \quad v^0_1 & v^0_1 \wedge t^0_{0,0} \Rightarrow z^0_{0,0,1} & t^0_{0,0} \Rightarrow v^0_1 \\
v^0_1 \Rightarrow (g^0_0 \vee g^0_1) & v^0_1 \wedge t^0_{1,1} \Rightarrow z^0_{0,0,1} & t^0_{1,1} \Rightarrow v^0_1 \\
z^0_{0,0,0} \Rightarrow \neg z^0_{0,1,0} & v^0_1 \wedge t^0_{1,0} \Rightarrow z^0_{1,0,1} & t^0_{1,0} \Rightarrow v^0_1 \\
z^0_{1,0,0} \Rightarrow \neg z^0_{1,1,0} & v^0_1 \wedge t^0_{1,2} \Rightarrow z^0_{1,0,1} & t^0_{1,2} \Rightarrow v^0_1 \\
v^0_1 \wedge g^0_0 \Leftrightarrow z^0_{0,0,1} & v^0_1 \wedge z^0_{0,0,1} \Rightarrow t^0_{0,0} \wedge t^0_{1,1} & v^0_1 \wedge g^0_0 \Rightarrow \neg g^0_1 \\
v^0_1 \wedge g^0_1 \Leftrightarrow z^0_{1,0,1} & v^0_1 \wedge z^0_{1,0,1} \Rightarrow t^0_{1,0} \wedge t^0_{2,1} & \\
v^1_{-1} \quad v^1_1 & v^1_1 \wedge t^1_{0,0} \Rightarrow z^1_{0,0,1} & t^1_{0,0} \Rightarrow v^1_1 \\
v^1_1 \Rightarrow (g^1_0 \vee g^1_1) & v^1_1 \wedge t^1_{1,1} \Rightarrow z^1_{0,0,1} & t^1_{1,1} \Rightarrow v^1_1 \\
z^1_{0,0,0} \Rightarrow \neg z^1_{0,1,0} & v^1_1 \wedge t^1_{1,0} \Rightarrow z^1_{1,0,1} & t^1_{1,0} \Rightarrow v^1_1 \\
z^1_{1,0,0} \Rightarrow \neg z^1_{1,1,0} & v^1_1 \wedge t^1_{1,2} \Rightarrow z^1_{1,0,1} & t^1_{1,2} \Rightarrow v^1_1 \\
v^1_1 \wedge g^1_0 \Leftrightarrow z^1_{0,0,1} & v^1_1 \wedge z^1_{0,0,1} \Rightarrow t^1_{0,0} \wedge t^1_{1,1} & v^1_1 \wedge g^1_0 \Rightarrow \neg g^1_1 \\
v^1_1 \wedge g^1_1 \Leftrightarrow z^1_{1,0,1} & v^1_1 \wedge z^1_{1,0,1} \Rightarrow t^1_{1,0} \wedge t^1_{2,1} &
\end{array}
$$

Y se agregan las siguientes cláusulas:

- Se deben cubrir todos los subobjetivos de Q usando alguna copia de la teoría *MCDs*.

$$
\begin{array}{c}
g^0_0 \vee g^1_0 \\
g^0_1 \vee g^1_1
\end{array}
$$

- Los subobjetivos cubiertos por las copias no se solapan.

$$
\begin{array}{c}
g^0_0 \Rightarrow \neg g^1_0 \\
g^0_1 \Rightarrow \neg g^1_1
\end{array}
$$

- Si se cubre el subobjetivo g_i con la copia t entonces la copia $t-1$ debe cubrir algún subobjetivo g_j con $j < i$.

$$
g^1_1 \Rightarrow \neg g^0_0
$$

Nótese que estas últimas cláusulas evitan que se consideren modelos que son simétricos. Por ejemplo, se evita producir la reescritura Q'', dado que ya se produce Q'.

$$
\begin{array}{c}
Q'(X,Z) : -v_0(X,Y), v_1(Y,Z) \\
Q''(X,Z) : -v_1(Y,Z), v_0(X,Y)
\end{array}
$$

Finalmente, la definición 5.2 permite obtener las reescrituras a partir de los modelos de la teoría extendida.

Definición 5.2 (Reescritura asociada a un modelo de la teoría extendida). Un modelo ω_E para la teoría T_E construida para $Q, \mathcal{V}$ representa un conjunto $\mathcal{C}_{\omega_E}$ de *MCDs* C_i no vacíos que cumplen la propiedad 2.

Considere la relación de equivalencia EC menos restrictiva consistente con todas las relaciones de equivalencias inducidas por φ_{C_i} para todo $C_i \in \mathcal{C}$. Se define ψ_i, sobre las variables de V_{C_i} de la siguiente manera:

$$\psi_i(x) = \begin{cases} [x] & x \in \mathit{Vars}(Q) \\ [\varphi_i^{-1}(x)] & x \in \mathit{Vars}(V_i) \text{ y } \varphi_i^{-1}(x) \neq \emptyset \\ x & x \in \mathit{Vars}(V_i) \text{ y } \varphi_i^{-1}(x) = \emptyset \end{cases}$$

donde $[y]$ corresponde al representante de la clase de equivalencia a la cual pertenece la variable y.

Luego, la reescritura asociada al conjunto de *MCDs* $\mathcal{C}$ es:

$$R(\psi(\vec{X})) :- V_{C_1}(\psi_1(h_{C_1}(\vec{Y}_1))), \cdots, V_{C_n}(\psi_n(h_{C_n}(\vec{Y}_n)))$$

El teorema 5.3 afirma que todo modelo de la teoría extendida corresponde a una reescritura válida para Q usando $\mathcal{V}$. La demostración de este teorema se presenta en el apéndice D.

Teorema 5.3. *Sea T_E la teoría extendida generada a partir de Q y $\mathcal{V}$. ω_E es un modelo para T_E si y solo si el conjunto $\mathcal{C}_{\omega_E}$ de MCDs obtenidos a partir de ω_E forman una reescritura válida para Q.*

Al igual, que para la generación de *MCDs*, el teorema 5.3 no garantiza que solamente se generen reescrituras minimales. Sin embargo, aunque se debe considerar MCDSAT como una aproximación al problema de reescrituras ya que no produce la solución exacta, el estudio experimental muestra la utilidad de esta solución sobre el algoritmo de *Minicon*. Además, se debe recordar de la sección 4.1, que *Minicon* tampoco garantiza conseguir la reescritura maximalmente contenida, siendo por esto una solución no exacta. Todo lo anterior se ilustra en el ejemplo 5.4.

Ejemplo 5.4. Las reescrituras que produce la teoría extendida para el siguiente problema de reescrituras:

$$\begin{aligned} Q(x,z) &:- \texttt{flight}(x,y), \texttt{flight}(y,z), \\ V_1(x_1,y_1,w_1) &:- \texttt{flight}(x_1,y_1), \texttt{flight}(y_1,z_1), \texttt{flight}(y_1,w_1) \end{aligned}$$

son las siguientes:

$$\begin{aligned} R_1(x,z) &:- V_1(x,y,z) \\ R_2(z,z) &:- V_1(y,z,y) \\ R_3(x,z) &:- V_1(x,y,w),\ V_1(y,z,v) \\ R_4(x,z) &:- V_1(w,x,y),\ V_1(v,y,z) \\ R_5(x,z) &:- V_1(x,y,w),\ V_1(v,y,z) \\ R_6(x,z) &:- V_1(y,z,w),\ V_1(v,x,y) \end{aligned}$$

Pero, $R_1 \subseteq R_5$ y $R_2 \subseteq R_6$, por lo tanto R_1, R_2 son redundantes. Estas reescrituras se produjeron al considerar los *MCDs* C_4, C_5 del ejemplo 5.2.

5.3. Implementación de la solución

En esta sección se presentan los aspectos más relevantes de la implementación de la solución propuesta en este trabajo, que llamaremos MCDSAT.

En la figura 5.1 se presentan los componentes de MCDSAT. A continuación se describen cada uno de ellos.

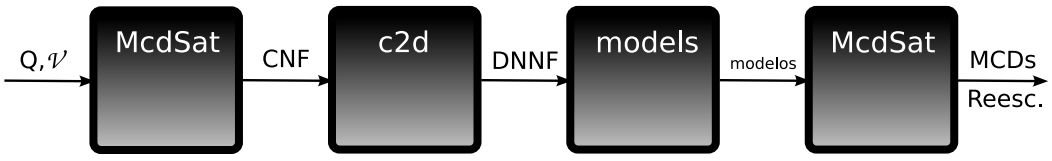

Figura 5.1: Componentes de MCDSAT.

5.3.1. Generación de la teoría lógica

En primer lugar, se traduce el problema de reescrituras en la teoría MCD/Extendida correspondiente usando las definiciones presentadas en los anexos A y C. Para ello, se implementó un generador que produce las cláusulas en CNF.

El generador recibe la definición de la consulta y el conjunto de vistas y el tipo de teoría se desea generar: MCD o extendida. En el apéndice E se presenta una descripción de los parámetros de entrada del generador de la teoría lógica.

La teoría se genera en el formato DIMACS [1]. Este formato es usado por la mayoría de los *SAT-solvers* y por el compilador d-DNNF que se usará en este trabajo.

A partir de la definición de las cláusulas, se deduce que la complejidad de la generación de las mismas es polinomial en la longitud de las consultas y las vistas y el número de vistas.

5.3.2. Compilador d-DNNF

Se utilizó el compilador `c2d` [Dar04] para transformar la teoría CNF a d-DNNF. Este compilador recibe la teoría CNF especificada en el formato DIMACS y produce la teoría compilada a d-DNNF. La descripción detallada del algoritmo empleado por este compilador se encuentra en [Dar04].

Se escogió utilizar este compilador debido a que sus autores mostraron en [Dar04] resultados muy superiores en comparación con el compilador a d-DNNF que se encontraba hasta el momento [Dar02]. Además, en [Dar02] se muestra que se ha logrado compilar en d-DNNF y contar modelos de una variedad de teorías CNF que no fueron posibles compilar en otros lenguajes de compilación, como OBDD [Bry86].

Sin embargo, si se desea, es posible cambiar el compilador y/o lenguaje de compilación. Incluso es posible, utilizar *SAT-solvers* para determinar si la teoría es satisfacible, es decir, si existe alguna reescritura o *MCD* para el problema.

[1] Una descripción de este formato se encuentra disponible en `www.satlib.org/Benchmarks/SAT/satformat.ps`

5.3.3. Obtención de modelos a partir del d-DNNF

Una vez generado el d-DNNF, los modelos se obtienen, a partir del d-DNNF, usando una herramienta desarrollada por el Prof. Blai Bonet, llamada MODELS.

Debido a que en esta aplicación, se pueden llegar a producir millones de modelos, no es útil hacer una implementación recursiva de la definición 4.9, por lo cual MODELS implementa el algoritmo iterativo que se describe en la sección 4.2.3.

La ventaja es que se pueden ir produciendo reescrituras a medida que se van produciendo los modelos. Además, es posible modificar MODELS para que obtenga solamente hasta una cierta cantidad de modelos y de esa manera limitar el número de *MCDs*/reescrituras que se desean calcular.

Si solamente se desea saber si el problema tiene reescrituras posibles, se puede cambiar este componente por un *SAT-solver*, como zChaff [2]. El *SAT-solver* solamente generaría un modelo, en el caso de que el problema tenga alguna reescritura posible.

5.3.4. Generación de *MCDs*/Reescrituras

Por último, se transforman los modelos obtenidos por MODELS, en *MCDs*/reescrituras del problema. Para ello se creó un programa que implementa la transformación descrita en las definiciones 5.1 y 5.2.

El programa recibe como entrada las definiciones de la consulta y las vistas y el conjunto de modelos de la teoría y produce los *MCDs*/reescrituras. En el apéndice E se describe la salida que genera la herramienta para describir los *MCDs*/reescrituras.

Al igual que el proceso de generación de cláusulas, la transformación de los modelos en *MCDs*/reescrituras es polinomial en el tamaño del problema y el número de modelos de la teoría.

[2] `http://www.princeton.edu/ chaff/zchaff.html`

Capítulo 6

Resultados experimentales

En este capítulo se reportan los resultados de un estudio experimental que tuvo como objetivo comparar el método propuesto en este trabajo, MCDSAT, y el algoritmo de *Minicon*.

No se compararon otros algoritmos, como *Bucket* o *Inverse rules*, debido a que en [PH01] se demuestra la superioridad del algoritmo de *Minicon*.

La sección 6.1 se dedica a explicar como se estructuraron los experimentos en el estudio comparativo. Luego, en la sección 6.2, se exponen los resultados obtenidos en cuanto a tiempo de ejecución y calidad de las respuestas. Finalmente, en la sección 6.3, se presentan las conclusiones derivadas del estudio experimental.

6.1. Configuración de los experimentos

El objetivo del estudio fue comparar las dos soluciones observando la influencia de los siguientes factores:

- El forma de las consultas y las vistas.
- El tamaño de las consultas y las vistas.
- El número de vistas.

Para el estudio, se implementó el algoritmo de *Minicon*[1], en el lenguaje Python. Además, se utilizó la implementación, en Java, de la primera fase de *Minicon* de [ALU01].

[1] No fue posible conseguir el código de la implementación de sus autores originales.

Los experimentos fueron ejecutados en un cluster de 24 máquinas con dos procesadores AMD de 2GHz y con 1Gb de memoria.

6.1.1. Generador de experimentos

Para realizar el análisis comparativo, se utilizó el generador de consultas utilizado en [ALU01] para realizar sus experimentos [2]. Este generador produce los subobjetivos de las consultas escogiendo aleatoriamente predicados de un conjunto de predicados disponibles. Las variables de los subobjetivos se generan siguiendo las restricciones de forma del tipo de consulta que se desee generar. Este generador, produce consultas de tres formas:

- Cadena: El subobjetivo j de la consulta comparte variables únicamente con los subobjetivos $j-1$ y $j+1$.
- Estrella: Hay un subobjetivo central que comparte variables con todos los demás subobjetivos de la consulta. Los otros subobjetivos únicamente comparten variables con el subobjetivo central.
- Aleatorios: No se impone ninguna restricción sobre las variables de la consulta.

Al generar las consultas se puede variar el número de subobjetivos, el número de predicados disponibles para los subobjetivos, el número de variables de los subobjetivos y el número de variables distinguibles. Los nombres de los predicados de los subobjetivos se generan aleatoriamente.

6.1.2. Estructura del estudio experimental

Se produjeron diferentes configuraciones para los experimentos. En cada configuración se varió el tipo de consulta, el tamaño de la consulta y las vistas, el número de vistas y el número de variables distinguibles en las consultas.

El tamaño de la consulta varía entre 3 y 10. El número de vistas se varió entre 10 y 150. El número de variables distinguibles puede ser la mitad o todas las variables. El número de predicados disponibles para la generación de los subobjetivos es 10.

[2] Comunicación personal con C. Li

Se ejecutaron dos experimentos para cada configuración. El primer experimento mide el tiempo de compilación de la teoría *MCD* y su número de modelos para MCDSAT y el tiempo para generar los *MCDs* y su número para el algoritmo de *Minicon*. En el segundo experimento se comparó el tiempo de compilar las teorías extendidas y su número de modelos y el tiempo de generar reescrituras y su número para el algoritmo de *Minicon*.

Además, se realizó un experimento con la finalidad de probar a MCDSAT con un problema que tiene un número exponencial de *MCDs* no minimales. Todos estos *MCDs* no minimales aparecen como modelos de la teoría pero no son generados por *Minicon*.

Se impuso una cota en tiempo y en memoria a cada corrida. Para los experimentos de generación de *MCDs* la cota en tiempo fue de 30 minutos, mientras que para los experimentos de generación de reescrituras fue de 2.5 horas. La cota en memoria fue de 1Gb para todos los experimentos.

En los experimentos se desprecia el tiempo de obtener los modelos y generar los *MCDs*/reescrituras, a partir de la teoría compilada.

6.2. Análisis de resultados

En esta sección se analizan los resultados obtenidos en el estudio experimental descrito en la sección anterior. Las gráficas se muestran en escala logarítmica. Cada punto en las gráficas representa el promedio de 10 evaluaciones de los algoritmos con la misma configuración. Los puntos que no se muestran en las gráficas corresponden a evaluaciones que no terminaron debido a la cota de tiempo o memoria.

Antes de presentar los resultados comparativos entre *Minicon* y MCDSAT, en la sección 6.2.1 se reporta el tamaño de las teorías CNF y DNNF generadas por MCDSAT, con el objetivo de medir la dificultad de generación y compilación de las mismas.

Posteriormente, en las secciones 6.2.2 y 6.2.3 se presenta el resultado del análisis comparativo entre *Minicon* y MCDSAT. El estudio compara el tiempo de ejecución de ambas soluciones y además analiza el efecto de la generación de *MCDs* no minimales en la calidad de las soluciones producidas por MCDSAT con respecto a las soluciones que produce *Minicon*. Para ver esto, las medidas de *Precision* y *Recall* se usan para comparar los *MCDs* producidos por ambos algoritmos. Estas medidas se definen como sigue:

Definición 6.1 (*Precision*). Relación entre el número de *MCDs*/reescrituras generados por *Minicon* entre el número de *MCDs*/reescrituras generados por MCDSAT.

$$Precision_M = \frac{|\text{ MCDs } Minicon \cap \text{MCDs MCDSAT}|}{|\text{ MCDs MCDSAT}|}$$
$$Precision_R = \frac{|\text{ Reesc. } Minicon \cap \text{Reesc. MCDSAT}|}{|\text{ Reesc. MCDSAT}|}$$

Definición 6.2 (*Recall*). Relación entre el número de *MCDs*/reescrituras generadas por MCDSAT que son correctas entre el número total de *MCDs*/reescrituras generadas.

$$Recall_M = \frac{|\text{MCDs MCDSAT} \cap \text{MCDs } Minicon|}{|\text{ MCDs } Minicon|}$$
$$Recall_R = \frac{|\text{Reesc. MCDSAT} \cap \text{Reesc. } Minicon|}{|\text{ Reesc. } Minicon|}$$

Por los teoremas 5.2 y 5.3, se garantiza que todos los *MCDs* y reescrituras generados por MCDSAT son correctos, es decir, $Recall_M = Recall_R = 1$.

Sin embargo, no hay garantía de que todo lo que genere MCDSAT es generado por *Minicon*, a menos que las vistas tengan todas las variables distinguibles. Es por esto que se desea estudiar las medidas $Precision_M$ y $Precision_R$ para todos los problemas con la mitad de las variables distinguibles. Por el teorema 5.2, para los problemas con todas las variables distinguibles $Precision_M = Precision_R = 1$.

6.2.1. Tamaños de las teorías generadas por MCDSAT

En la tabla 6.1 se muestra el tamaño promedio de los CNF (medido en número de variables y cláusulas) y los DNNF (medido en número de nodos y arcos) generados por MCDSAT para calcular *MCDs* en problemas con la mitad de las variables distinguibles.

Se observa que las teorías para consultas estrella son más grandes que las teorías para consultas cadena y aleatorias, tanto las CNF como las d-DNNF.

En las secciones 6.2.2 y 6.2.3 se muestra el tiempo de generación y compilación de estas teorías. No necesariamente el tamaño de la teoría CNF es proporcional a su tiempo de compilación y/o al tamaño del d-DNNF resultante. Se debe notar que si la teoría no es satisfacible (no existe ningún *MCD*/reescritura) el d-DNNF tiene un solo nodo.

Problema		CNF		d-DNNF	
tipo	# subobs	# vars.	# claus.	# nodos	# arcos
Cadena	3	190.6	4595.4	651.0	5642.6
	6	502.9	8209.7	2191.8	29333.4
	9	998.5	14052.3	4590.0	66113.1
Estrella	3	178.7	4422.7	621.9	4887.1
	6	577.2	9381.8	2309.9	36048.3
	9	1440.0	20981.1	5459.0	110153.2
Aleatorio	3	185.0	4703.1	646.7	5182.3
	6	394.0	8066.7	1913.1	21215.7
	9	765.0	13524.8	4125.6	46924.1
Cadena	3	571.8	13803.2	1080.3	7463.8
	6	3017.4	49380.2	5047.8	63992.3
	9	8986.5	126868.7	13342.1	229463.2
Estrella	3	536.1	13285.1	1025.0	6709.5
	6	3463.2	56412.8	5513.8	77583.8
	9	12960.0	189227.9	17694.2	392604.5
Aleatorio	3	555.0	14126.3	1092.1	7536.0
	6	2364.0	48522.2	4246.5	42200.4
	9	6885.0	122121.2	11071.7	140806.4

Tabla 6.1: Tamaño promedio de las teorías *MCD* (arriba) y extendida (abajo) para problemas con 80 vistas y la mitad de las variables distinguibles.

6.2.2. Resultados para generación de *MCDs*

Consultas tipo cadena

La figura 6.1 muestra los tiempos de ejecución para los experimentos de consultas y vistas tipo cadena. Las gráficas superiores muestran resultados para consultas con la mitad de las variables distinguibles, las inferiores muestran resultados para consultas con todas las variables distinguibles. En (a) y (c) se grafica el tiempo de ejecución versus el número de vistas. Se usaron consultas y vistas de tamaño 8 variando el número de vistas. En (b) y (d) se gráfica el tiempo de ejecución versus el tamaño de las consultas. Se usaron 80 vistas variando el tamaño de las consultas.

En primer lugar, se analizan los tiempos para consultas con la mitad de las variables distinguibles. En la gráfica (a) se observa que el tiempo del algoritmo de *Minicon* es menor para consultas de tamaño 3, 4 y 5. Sin embargo, para consultas de mayor tamaño, la compilación de la teoría se realiza en un tiempo mucho menor. Además, se observa que aumentar el tamaño de las consultas tiene mayor impacto en el tiempo de ejecución que el número de vistas para el algoritmo de

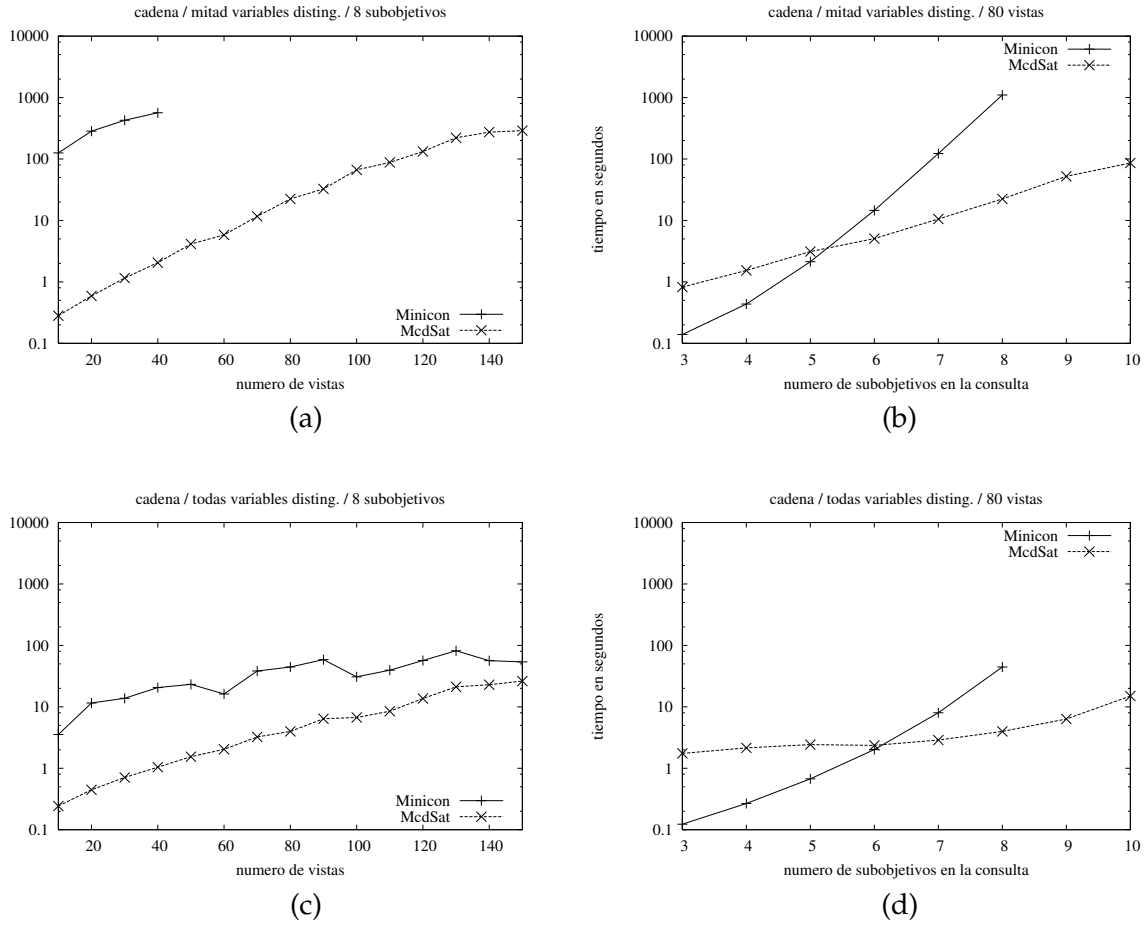

Figura 6.1: Comparación del tiempo de ejecución de la primera fase de *Minicon* y compilación de la teoría MCD para consultas tipo cadena.

Minicon. Por otro lado, en la gráfica (b), se puede observar que el tiempo requerido por MCDSAT es mucho menor que para el algoritmo de *Minicon* en todas las instancias.

Se observan resultados similares para consultas con todas las variables distinguibles. Para instancias pequeñas, el algoritmo de *Minicon* presenta un mejor tiempo de ejecución, pero al aumentar el número de subobjetivos en la consulta, MCDSAT escala mucho mejor.

La tabla 6.2 muestra el número promedio de *MCDs* generados por *Minicon* y MCDSAT para problemas de consultas de tipo cadena y 80 vistas. Se observó que al aumentar el número de subobjetivos en las consultas aumenta la cantidad de *MCDs* no minimales generados por MCDSAT, como era de esperarse.

Problema		# *MCDs*		
vars dist.	# subobs	*Minicon*	MCDSAT	$Precision_M$
Todas	3	74.6	74.6	1
	4	125.5	125.5	1
	5	200.9	200.9	1
	6	296.9	296.9	1
	7	398.9	398.9	1
	8	510.2	510.2	1
	9	-	648.5	1
	10	-	807.2	1
Mitad	3	11.2	11.5	0.97
	4	22.4	23.7	0.94
	5	33.8	38.8	0.87
	6	47.2	55.6	0.84
	7	62.3	78.4	0.79
	8	85.2	122.6	0.69
	9	-	164.4	-
	10	-	207.9	-

Tabla 6.2: Número promedio de *MCDs* generados por *Minicon* y MCDSAT para problemas con consultas tipo cadena y 80 vistas.

Es importante notar que para instancias del mismo tamaño, los problemas con todas las variables distinguibles toman menos tiempo y producen mayor cantidad de *MCDs* que los problemas con la mitad de las variables distinguibles. La razón es que el algoritmo de *Minicon* debe hacer menos comparaciones ya que los subobjetivos de las vistas, al tener todas las variables distinguibles, pueden cubrir por si solos a los subobjetivos de la consulta. En el caso de MCDSAT, las cláusulas de minimización reducen la cantidad modelos de las teorías lo cual puede influir en el tiempo de compilación.

Consultas tipo estrella

La figura 6.2 muestra los tiempos de ejecución para los experimentos con consultas de tipo estrella. Las gráficas superiores presentan los resultados para consultas con la mitad de las variables distinguibles y las inferiores presentan los resultados para consultas con todas las variables distinguibles. En (a) y (c) se presenta el tiempo de ejecución versus el número de vistas para consultas y vistas de tamaño 8 variando el número de vistas. En (b) y (d) se grafica el tiempo de ejecución versus el tamaño de las consultas en experimentos con 80 vistas variando el tamaño de las consultas.

Se observó el mismo comportamiento que para los experimentos tipo cadena, tanto para *Minicon* como para MCDSAT, aunque aumentó el tiempo de compilación de la teoría en relación con las

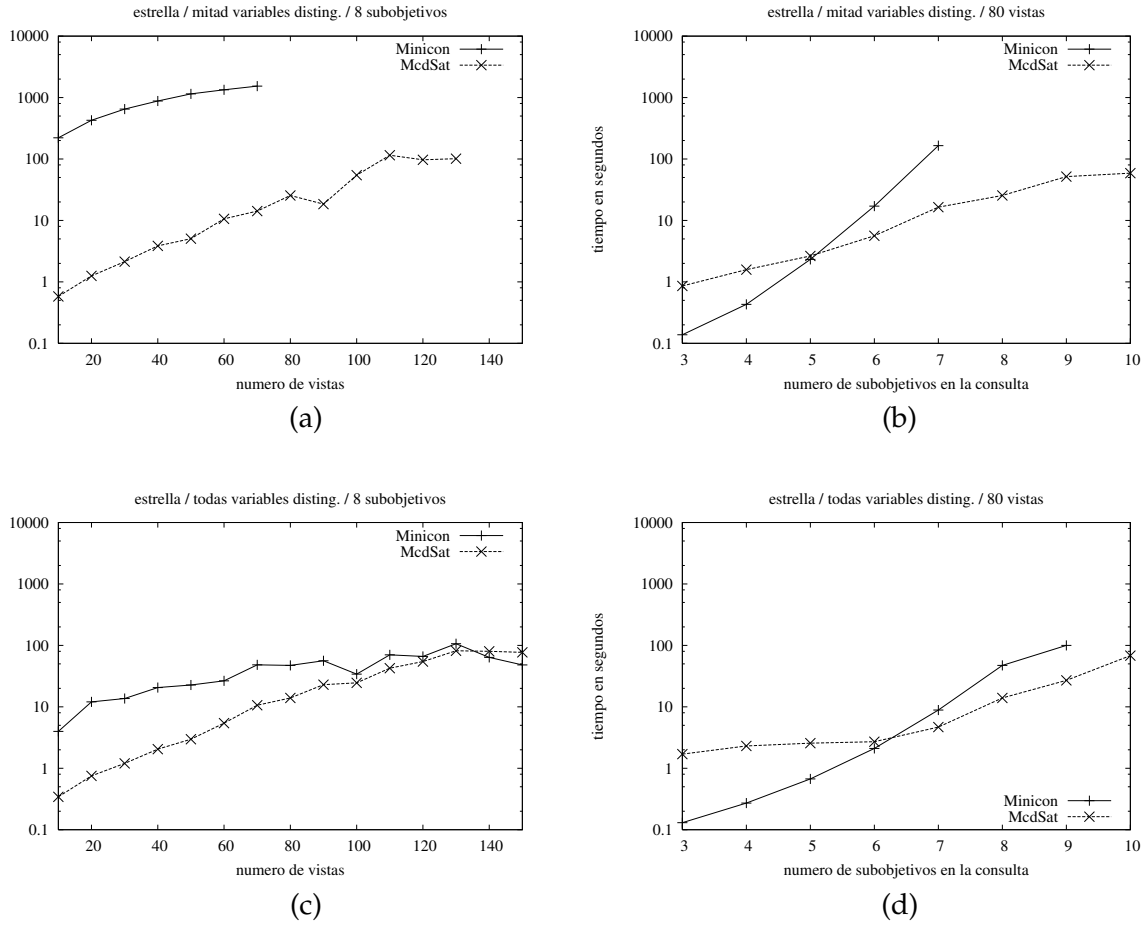

Figura 6.2: Comparación del tiempo de ejecución de la primera fase de *Minicon* y compilación de la teoría MCD para consultas tipo estrella.

consultas de tipo cadena. Esto pudo deberse a que el generador de consultas, para las consultas tipo estrella, genera muchas más variables que para las consultas tipo cadena, lo que hace que la teoría producida por MCDSAT sea mucho más grande, como puede verse en la tabla 6.3.

Sin embargo, para problemas con todas las variables distinguibles, *Minicon* muestra mejor tiempo que MCDSAT cuando la cantidad de vistas es más grande. Esto tiene la misma explicación que para consultas cadena. *Minicon* tiene que hacer menor cantidad de comparaciones cuando las vistas tienen todas las variables distinguibles.

La tabla 6.3 muestra el número promedio de *MCDs* generados por *Minicon* y MCDSAT para problemas de consultas de tipo estrella y 80 vistas. Se observó un comportamiento similar de MCDSAT, en cuanto a *MCDs* no minimales, con respecto a las consultas cadena.

Problema		# *MCDs*		
vars dist.	# subobs	*Minicon*	MCDSAT	$Precision_M$
Todas	3	73.7	73.7	1
	4	127.8	127.8	1
	5	196.5	196.5	1
	6	289.2	289.2	1
	7	398.0	398.0	1
	8	513.9	513.9	1
	9	645.0	645.0	1
	10	-	807.3	1
Mitad	3	19.3	19.6	0.98
	4	20.1	20.5	0.98
	5	24.0	26.2	0.91
	6	29.3	30.3	0.96
	7	28.2	31.8	0.88
	8	-	25.8	-
	9	-	26.3	-
	10	-	26.8	-

Tabla 6.3: Número promedio de *MCDs* generados por *Minicon* y MCDSAT para problemas con consultas tipo estrella y 80 vistas.

Consultas aleatorias

La figura 6.3 muestra los tiempos de ejecución para los experimentos con consultas sin restricción de forma. Las gráficas superiores presentan los resultados para consultas con la mitad de las variables distinguibles y las inferiores presentan los resultados para consultas con todas las variables distinguibles. En (a) y (c) se presenta el tiempo de ejecución versus el número de vistas. Las consultas y vistas de tamaño 8 variando el número de vistas. En (b) y (d) se grafica el tiempo de ejecución versus el tamaño de las consultas. Se usaron 80 vistas en cada experimento, variando el tamaño de las consultas.

En todos los experimentos se observó tanto para los experimentos tipo cadena y estrella, tanto para *Minicon* como para la compilación de la teoría. *Minicon* muestra el peor tiempo de ejecución de las dos soluciones. Además, los experimentos con todas las variables distinguibles presentaron un tiempo de ejecución menor con respecto a los experimentos con la mitad de las variables distinguibles.

Igualmente, disminuyó el tiempo de compilación de la teoría y *Minicon* en relación con las consultas de tipo cadena y estrella.

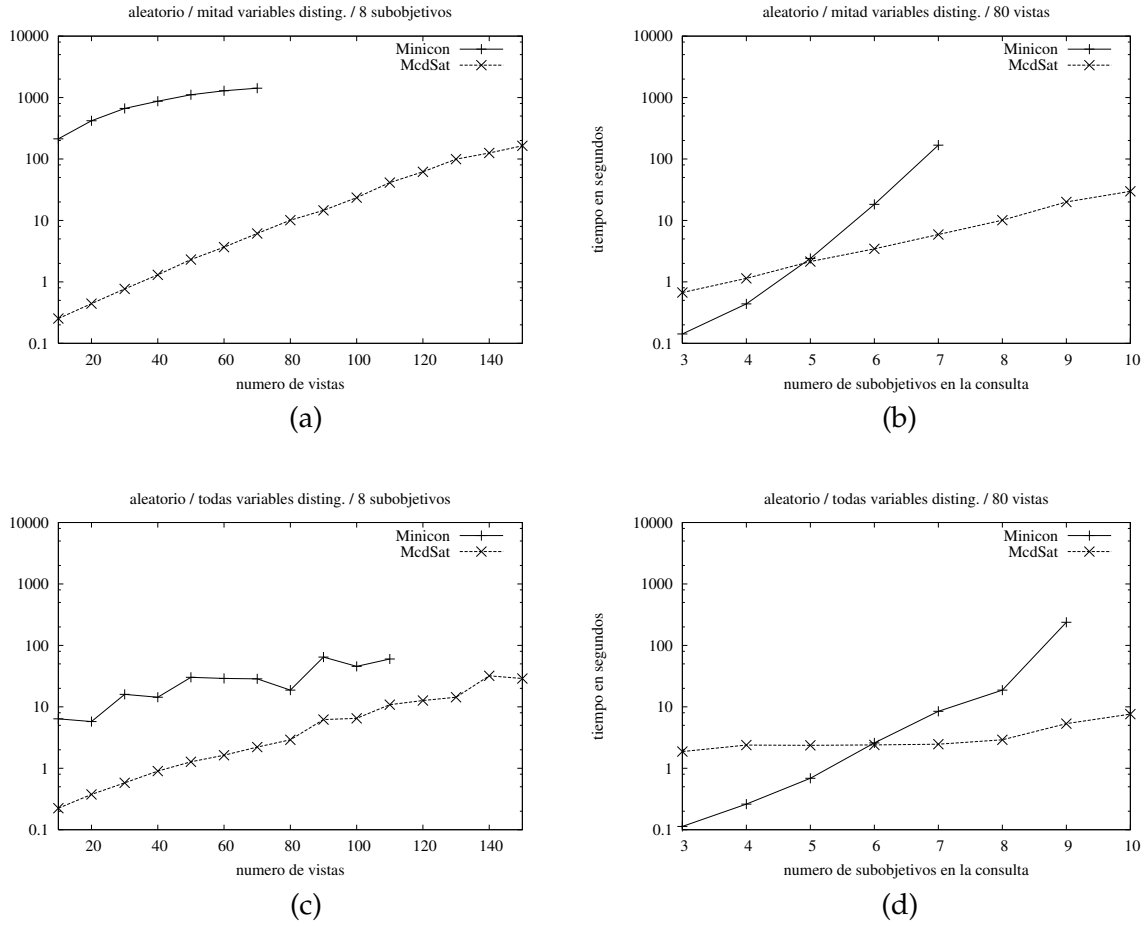

Figura 6.3: Comparación del tiempo de ejecución de la primera fase de *Minicon* y compilación de la teoría MCD para consultas aleatorias.

Por último, la tabla 6.4 muestra el número promedio de *MCDs* generados por *Minicon* y MCDSAT para problemas de consultas de aleatorias y 80 vistas. Se observó un comportamiento similar de MCDSAT, en cuanto a *MCDs* no minimales, con respecto a las consultas cadena y estrella, aunque se observa que se generan menor cantidad de *MCDs*.

Problema		# *MCDs*		
vars dist.	# subobs	*Minicon*	MCDSAT	$Precision_M$
Todas	3	77.0	77.0	1
	4	123.4	123.4	1
	5	197.0	197.0	1
	6	282.0	282.0	1
	7	393.0	393.0	1
	8	512.1	512.1	1
	9	650.0	650.0	1
	10	795.9	795.9	1
Mitad	3	5.1	5.1	1
	4	9.1	9.2	0.98
	5	15.9	16.7	0.95
	6	21.9	24.4	0.89
	7	29.3	34.6	0.84
	8	-	49.7	-
	9	-	58.0	-
	10	-	87.5	-

Tabla 6.4: Número promedio de *MCDs* generados por *Minicon* y MCDSAT para problemas con consultas aleatorias y 80 vistas.

Experimentos con número exponencial de *MCDs*

El problema de reescrituras, con número exponencial de *MCDs* se define de la siguiente manera:

$$\begin{aligned} Q(x,y) &:- p_1(x,z_1),\, q_1(z_1,y), \ldots, p_k(x,z_k),\, q_k(z_k,y)\,, \\ V_1(x,y) &:- p_1(x,z_1),\, q_1(z_1,y)\,, \\ &\vdots \\ V_k(x,y) &:- p_1(x,z_1),\, q_1(z_1,y), \ldots, p_k(x,z_k),\, q_k(z_k,y)\,. \end{aligned}$$

Es posible demostrar que *Minicon* calcula solo $k(k+1)/2$ *MCDs* minimales, mientras que MCDSAT produce todos los *MCDs*, es decir, $2^{k+1} - k - 2$ *MCDs*. Los resultados se muestran en la tabla 6.5. Sin embargo, incluso en este problema de reescrituras, MCDSAT es capaz de generar los *MCDs* en menos tiempo que *Minicon*, debido a que este último necesita generar y probar todos los *MCDs*.

	Minicon		MCDSAT	
k	# MCDs	tiempo	# MCDs	tiempo
4	10	0.6	26	0.0
6	21	1.6	120	0.1
8	36	11.2	502	0.2
10	55	80.1	2,036	0.4

Tabla 6.5: Resultados para el problema con exponencial número de *MCDs*.

6.2.3. Resultados para generación de reescrituras

Sobre el mismo conjunto de datos, presentados en la sección anterior, se ejecutaron experimentos para la generación de reescrituras. A continuación se presentan los resultados.

Consultas tipo cadena

La figura 6.4 muestra los tiempos de ejecución para los experimentos de consultas y vistas tipo cadena. Las gráficas superiores presentan los resultados para consultas con la mitad de las variables distinguibles y las inferiores presentan los resultados para consultas con todas las variables distinguibles. En (a) y (c) se presenta el tiempo de ejecución versus el número de vistas para experimentos con consultas y vistas de tamaño 6 variando el número de vistas. En (b) y (d) se grafica el tiempo de ejecución versus el tamaño de las consultas para experimentos con 80 vistas aumentando el tamaño de las consultas.

Como era de esperarse, aumenta el tiempo de generación de la teoría extendida en comparación con la teoría MCD. Notar que para los experimentos con todas las variables distinguibles, *Minicon* no fue capaz de terminar dentro de los límites establecidos ni siquiera para las instancias más pequeñas.

Al igual que para los experimentos de generación de *MCDs*, el tiempo de ejecución de *Minicon* es mayor al tiempo de compilación de la teoría en todos los experimentos. También se observa que aumentar el tamaño de las consultas tiene mayor impacto en el tiempo de ejecución que el número de vistas tanto para MCDSAT como para el algoritmo de *Minicon*.

La tabla 6.6 muestra el número promedio de reescrituras generadas por *Minicon* y MCDSAT para problemas de consultas cadena con 80 vistas.

Es importante ver que, contrario a lo que se observó para la generación de *MCDs*, los problemas con todas las variables distinguibles tardan más tiempo en resolverse que los problemas con la

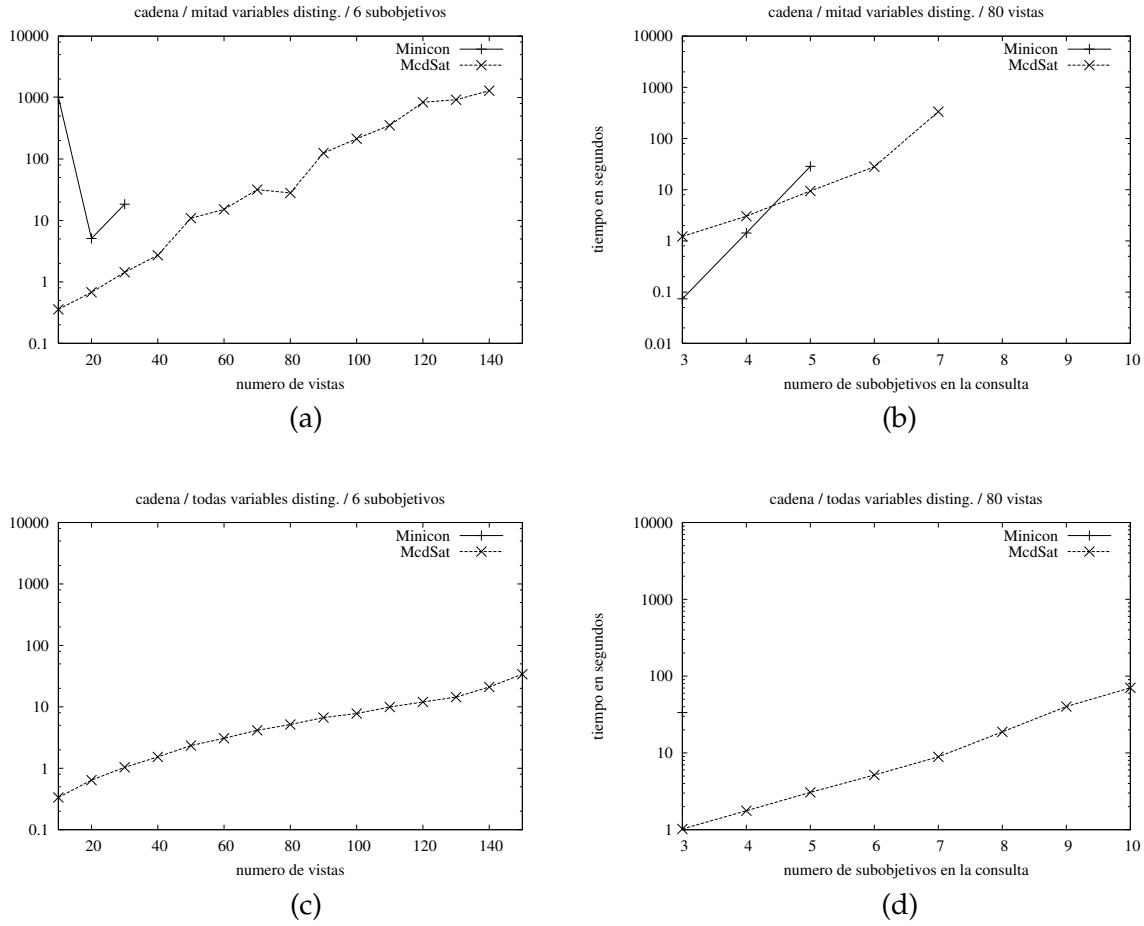

Figura 6.4: Comparación del tiempo de ejecución del algoritmo de *Minicon* y compilación de la teoría extendida para consultas cadena.

mitad de las variables distinguibles. Esto se debe a que hay mayor cantidad de *MCDs* cuando todas las variables son distinguibles y la fase de combinación de *Minicon* se hace más costosa.

Se observa que para los problemas con todas las variables distinguibles, son capaces de producir millones de reescrituras. Aunque todas estas reescrituras serían imposibles de evaluar en un sistema real, MCDSAT es capaz de evaluar cual es la cantidad de reescrituras posibles, antes de producirlas. Además, es posible modificar MCDSAT para que solo produzca una fracción de las reescrituras a partir de la teoría compilada.

Problema		# *MCDs*		
vars dist.	# subobs	*Minicon*	MCDSAT	$Precision_R$
Todas	3	14907.4	14907,4	1
	4	-	960093,4	1
	5	-	$1{,}03 \times 10^8$	1
	6	-	$1{,}46 \times 10^{10}$	1
	7	-	$1{,}95 \times 10^{12}$	1
	8	-	$2{,}68 \times 10^{14}$	1
	9	-	$5{,}09 \times 10^{16}$	1
	10	-	$1{,}14 \times 10^{19}$	1
Mitad	3	29.2	29,7	0.98
	4	524.2	543,0	0.96
	5	4843.1	5247,2	0.92
	6	-	110240,4	-
	7	-	$1{,}68 \times 10^6$	-
	8	-	-	-
	9	-	-	-
	10	-	-	-

Tabla 6.6: Número promedio de reescrituras generadas por *Minicon* y MCDSAT para problemas con consultas tipo cadena y 80 vistas.

Consultas tipo estrella

La figura 6.5 muestra los tiempos de ejecución para los experimentos de consultas y vistas tipo estrella. Las gráficas superiores presentan los resultados para consultas con la mitad de las variables distinguibles y las inferiores presentan los resultados para consultas con todas las variables distinguibles. En (a) y (c) se presenta el tiempo de ejecución versus el número de vistas. Las consultas y vistas de tamaño 6 variando el número de vistas. En (b) y (d) se muestra el tiempo de ejecución versus el tamaño de las consultas. En cada experimento, se usaron 80 vistas variando el tamaño de las consultas.

En general, se observa el mismo comportamiento que para las consultas cadena. MCDSAT exhibe un mejor tiempo de ejecución que *Minicon* para las instancias más grandes.

La tabla 6.7 muestra el número promedio de reescrituras generadas por *Minicon* y MCDSAT para problemas de consultas estrella y 80 vistas. Obsérvese que para problemas en los cuáles MCDSAT no produce reescrituras, se puede asegurar que $Precision_R = 1$. Además, para problemas en los que todas las variables son distinguibles se produjeron mayor cantidad de reescrituras, en ambas soluciones. Esto es consecuencia de lo que se observó en la sección 6.2.2. Los experimentos con todas las variables distinguibles generan mayor cantidad de *MCDs*, por lo cual hay mayor cantidad de reescrituras.

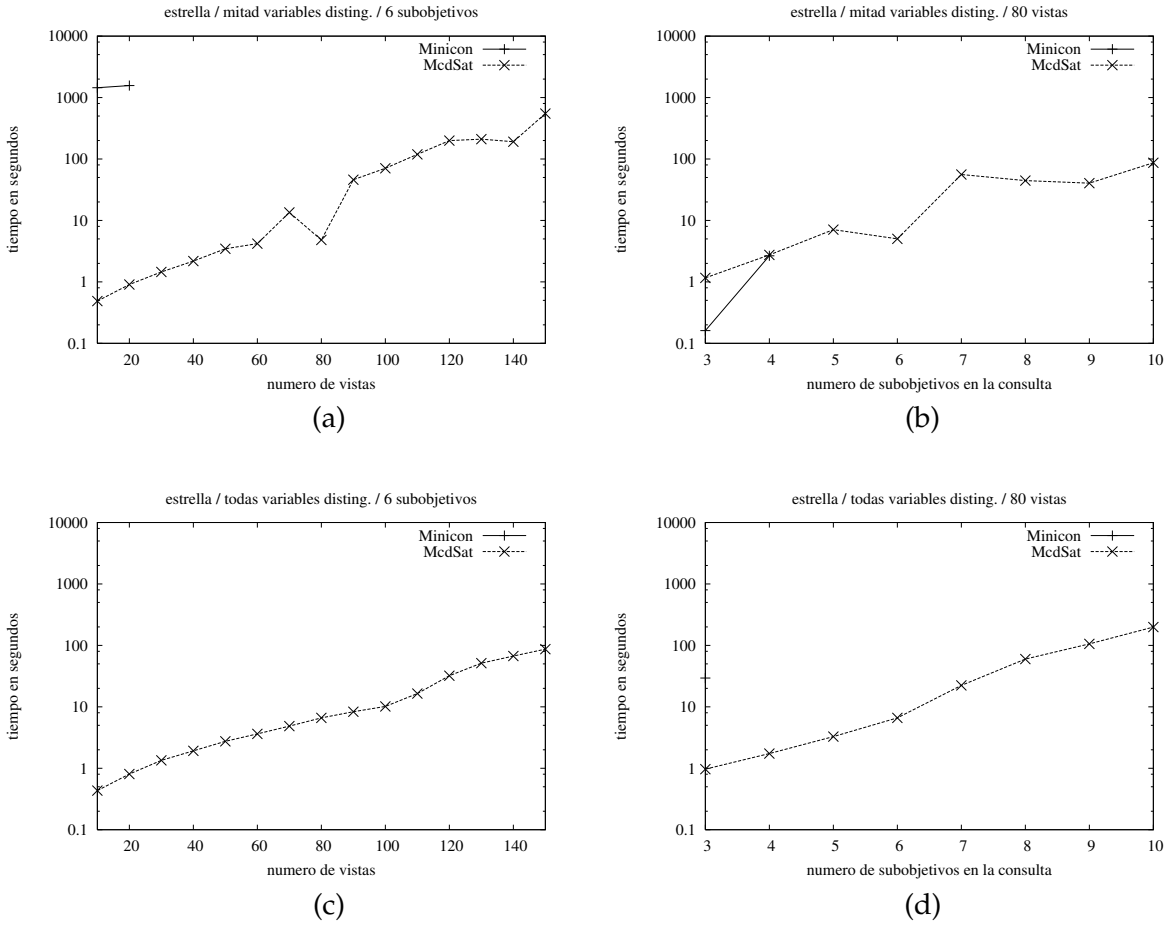

Figura 6.5: Comparación del tiempo de ejecución del algoritmo de *Minicon* y compilación de la teoría extendida para consultas estrella.

Consultas tipo aleatorias

La figura 6.6 muestra los tiempos de ejecución para los experimentos de consultas y vistas sin restricciones de forma. Las gráficas superiores presentan los resultados para consultas con la mitad de las variables distinguibles y las inferiores presentan los resultados para consultas con todas las variables distinguibles. En (a) y (c) se presenta el tiempo de ejecución versus el número de vistas. Las consultas y vistas de tamaño 6 variando el número de vistas. En (b) y (d) se grafica el tiempo de ejecución versus el tamaño de las consultas. Se usaron 80 vistas en cada experimento, variando el tamaño de las consultas.

Para el caso de consultas aleatorias con la mitad de las variables distinguibles, se obtuvo mejor comportamiento para *Minicon* que MCDSAT. Aunque para instancias de longitud mayor que 7,

Problema		# *MCDs*		
vars dist.	# subobs	*Minicon*	MCDSAT	$Precision_R$
Todas	3	14772.4	14772.4	1
	4	-	1041639.0	1
	5	-	$9{,}0 \times 10^7$	1
	6	-	$1{,}2 \times 10^{10}$	1
	7	-	$1{,}8 \times 10^{12}$	1
	8	-	$2{,}8 \times 10^{14}$	1
	9	-	$4{,}8 \times 10^{16}$	1
	10	-	$1{,}1 \times 10^{19}$	1
Mitad	3	149.6	156.0	0.958974
	4	362.7	431.9	0.839778
	5	-	1342.7	-
	6	-	0.0	1
	7	-	2196.0	-
	8	-	249.6	-
	9	-	0.0	1
	10	-	0.0	1

Tabla 6.7: Número promedio de reescrituras generadas por *Minicon* y MCDSAT para problemas con consultas tipo estrella y 80 vistas.

Minicon no fue capaz de terminar dentro de las cotas impuestas.

En los experimentos con todas las variables distinguibles, se observó el mismo comportamiento que para experimentos con consultas cadena y estrella. *Minicon* no fue capaz de producir respuesta, dentro de las cotas de ejecución impuestas, ni siquiera para las instancias más pequeñas.

La tabla 6.8 muestra el número promedio de reescrituras generadas por *Minicon* y MCDSAT para problemas de consultas aleatorias y 80 vistas.

6.3. Conclusiones del estudio experimental

En el estudio experimental se pudo mostrar lo siguiente:

- MCDSAT escala mejor que el algoritmo de *Minicon* para generar *MCDs* y reescrituras tanto al aumentar número de vistas como al aumentar el tamaño de la consultas, en la mayoría de los experimentos ejecutados. Este era uno de los objetivos principales de este trabajo, conseguir una solución más rápida que las soluciones existentes.

  Una parte de la mejora que muestra MCDSAT con respecto a *Minicon* puede explicarse en el

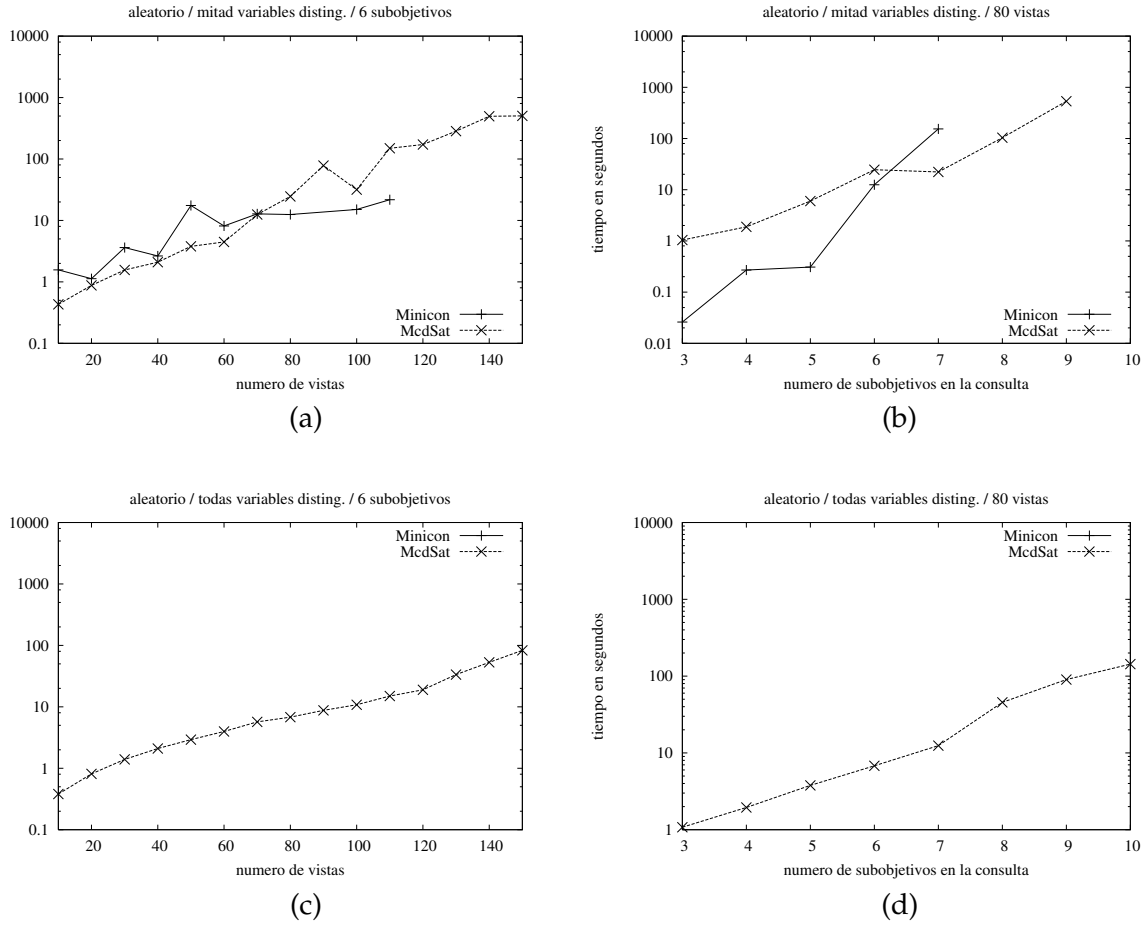

Figura 6.6: Comparación del tiempo de ejecución del algoritmo de *Minicon* y compilación de la teoría extendida para consultas aleatorias.

hecho que *Minicon* está implementado en Java/Python mientras que el compilador está implementado en C++. Sin embargo, la diferencia que se observa entre *Minicon* y MCDSAT es exponencial, por lo cual no puede acreditarse solamente a una diferencia en el lenguaje de implementación de las soluciones.

Esta mejora significativa se explica más bien en las técnicas que usa el compilador para hacer la transformación a d-DNNF. Estas técnicas fueron descritas en el capítulo 4, cuando se describió el compilador 2̧d.

- Aunque MCDSAT no es capaz de conseguir la reescritura maximalmente contenida en todas las instancias de los experimentos ejecutados, se mostró su utilidad en ciertas situaciones:

  - Cuando todas las variables son distinguibles, MCDSAT produce exactamente la misma

Problema		# *MCDs*		
vars dist.	# subobs	*Minicon*	MCDSAT	$Precision_R$
Todas	3	14801.4	14801.4	1
	4	-	956674.9	-
	5	-	$1{,}0 \times 10^8$	-
	6	-	$1{,}1 \times 10^{10}$	-
	7	-	$1{,}7 \times 10^{12}$	-
	8	-	$2{,}8 \times 10^{14}$	-
	9	-	$5{,}1 \times 10^{16}$	-
	10	-	$1{,}0 \times 10^{19}$	-
Mitad	3	3.0	3.0	1
	4	2.4	2.4	1
	5	12.4	14.4	0.861111
	6	151.2	167.4	0.903226
	7	230.4	243.6	0.945813
	8	-	1659.3	-
	9	-	37696.7	-
	10	-	-	-

Tabla 6.8: Número promedio de reescrituras generadas por *Minicon* y MCDSAT para problemas con consultas tipo estrella y 80 vistas.

solución que *Minicon*.

- MCDSAT es capaz de producir reescrituras en problemas para los cuales *Minicon* no es capaz de responder.
- Es posible decidir si un problema tiene reescrituras posibles usando MCDSAT en instancias muy grandes. En los casos en los que existan demasiadas reescrituras, sería imposible evaluarlas todas. Sin embargo, MCDSAT sería capaz de calcular la cantidad total de reescrituras y solo producir una fracción de las mismas.

- La forma de las consultas influyen en el tiempo de ejecución tanto del algoritmo de *Minicon* como MCDSAT.

  Las consultas estrella, generadas en los experimentos, poseían un mayor número de variables lo que incrementó el tamaño de las teorías para generar *MCDs*e influyó en el tiempo de ejecución los experimentos.

  Sin embargo, el tiempo de compilación de las teorías extendidas para consultas estrella fue menor que en las consultas tipo cadena debido a que, por su forma, se obtienen muy pocas reescrituras con consultas estrella.

  Los experimentos para consultas aleatorias fueron los que tuvieron menor tiempo de ejecución, tanto para *Minicon* como para MCDSAT. Esto se debió a que las consultas aleatorias, producidas por el generador, producían menor cantidad de *MCDs*.

- El tamaño de las consultas y las vistas influye más en el tiempo de ejecución de ambas soluciones que el número de vistas. La razón es que la longitud de todas las vistas es la misma, por lo que al aumentar la longitud de las mismas crece más la cantidad de subobjetivos con los que se puede cubrir la consulta.

- El número de variables distinguibles influyó en el tiempo de ejecución de ambas soluciones.

  En el caso de los experimentos para generación de *MCDs*, las consultas con la mitad de las variables distinguibles tuvieron mayor tiempo de ejecución. En el algoritmo de *Minicon*, la fase de generación de *MCDs* es más sencilla cuando todas las variables son distinguibles ya que los subobjetivos de la vista pueden cubrir por sí solos cualquier subobjetivo consistente de la consulta. En MCDSAT, las cláusulas de minimización reducen la cantidad de *MCDs* producidos por la teoría.

  Por el contrario, en el caso de la generación de reescrituras, los experimentos con todas las variables distinguibles tuvieron el mayor tiempo de ejecución. Esto se debe a que cuando todas las variables son distinguibles se genera mayor cantidad de *MCDs*, por lo cual la fase de combinación es más costosa.

Capítulo 7

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones alcanzadas luego del desarrollo de este trabajo.

En la sección 7.1 se presentan las conclusiones y finalmente, en la sección 7.2, se dan posibles extensiones para este trabajo.

7.1. Conclusiones

Entre las principales conclusiones de este trabajo, se encuentran las siguientes:

- Se desarrolló una nueva solución para el problema de reescritura de consulta usando vistas. Este problema es de gran importancia para sistemas de integración de datos, debido a que este tipo de sistemas necesitan reformular las consultas, construidas sobre su esquema global, para poder referirse a las vistas.
- El nuevo enfoque propuesto, MCDSAT, basado en el concepto de *MCD* del algoritmo de *Minicon*, consistió en la construcción de una teoría lógica tal que sus modelos correspondieran a los *MCDs* del problema. Luego, las soluciones podrían obtenerse a partir de estos *MCDs* o extendiendo la teoría para que sus modelos correspondieran a las reescrituras del problema.

  Aunque el nuevo enfoque no consigue dar exactamente la reescritura maximalmente contenida del problema, se mostró la utilidad del mismo en algunas situaciones.
- El uso de una representación lógica, permitió hacer uso de los nuevos desarrollos en el área de compilación de conocimiento y SAT, los cuales son capaces de tratar con teorías de miles de variables y cláusulas.

- La transformación del problema a lógica es bastante sencilla, utilizando las definiciones y propiedades de los *MCD*. Por el contrario, la implementación del algoritmo de *Minicon* no es directa a partir de las definiciones y requiere hacer algunas suposiciones sobre las mismas.
- El estudio experimental mostró mejoría en el tiempo de ejecución de MCDSAT en comparación con el algoritmo de *Minicon*, tanto para la generación de *MCDs* como para la generación de reescrituras. Estos resultados se mostraron para una gran variedad de problemas sintetizados, variando la forma y el tamaño de las consultas, la cantidad de variables distinguibles y la cantidad de vistas.

7.2. Trabajo Futuro

A continuación se enumeran las posibles extensiones de este trabajo:

- La extensión más directa de este trabajo consiste en permitir la presencia de constantes en el cuerpo de la consulta y las vistas. Esta extensión es bastante sencilla, ya que consiste en modelar las cláusulas lógicas para las restricciones que se definen en [PH01] para considerar constantes en el algoritmo de *Minicon*.
- Es posible extender la teoría lógica que representa a las reescrituras, de manera que pueda ser usada para responder diferentes consultas que cumplan ciertas restricciones. Por ejemplo, agregando nuevas variables y cláusulas proposicionales sería posible construir y compilar una teoría general que pueda responder consultas de tamaño N.

  Luego, para reescribir una determinada consulta, que cumpla estas condiciones, sería necesario instanciar ciertas variables proposicionales de manera que los modelos de la teoría sean reescrituras válidas para esa consulta. Aunque las teorías generadas de esta manera serían mas grandes y difíciles de compilar, este tiempo de compilación sería amortizado en responder diferentes consultas.
- Los experimentos mostrados en este trabajo fueron completamente sintetizados usando un generador de consultas aleatorio. Pero para comprobar la utilidad de la nueva solución, es necesario hacer experimentos en problemas reales.

  Además, es posible ampliar el estudio experimental haciendo uso de otros lenguajes de compilación y/o cambiando los parámetros de compilación que usa la herramienta actualmente, con la idea de mejorar el tiempo de ejecución de MCDSAT.

- Utilizando como ejemplo la transformación lógica del problema de reescrituras presentado en este trabajo, se puede modelar otro tipo de problemas y hacer uso de las mismas técnicas de representación de conocimiento usados en este trabajo.

  Los resultados obtenidos en este trabajo muestran la utilidad de las técnicas de compilación de conocimiento para resolver este tipo de problemas.

Bibliografía

[ABV06] Yolifé Arvelo, Blai Bonet, and Maria-Esther Vidal. Compilation of query-rewriting problems into tractable fragments of propositional logic. In *Proceedings of The Twenty-First National Conference on Artificial Intelligence, AAAI'06, and the Eighteenth Innovative Applications of Artificial Intelligence Conference, IAAI'06, July 16-20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006.

[ALU01] Foto Afrati, Chen Li, and Jeffrey Ullman. Generating efficient plans for queries using views. In *SIGMOD 2001 Electronic Proceedings*, pages 319–330, 2001.

[BDD+98] Randall G. Bello, Karl Dias, Alan Downing, James J. Feenan Jr., James L. Finnerty, William D. Norcott, Harry Sun, Andrew Witkowski, and Mohamed Ziauddin. Materialized views in oracle. In *Proceedings of 24rd International Conference on Very Large Data Bases, VLDB'98, August 24-27, 1998, New York City, New York, USA*, pages 659–664. Morgan Kaufmann, 1998.

[BGRV99] Laura Bright, Jean-Robert Gruser, Louiqa Raschid, and Maria Esther Vidal. A wrapper generation toolkit to specify and construct wrappers for Web accessible data sources (WebSources). *International Journal of Computer Systems Science and Engineering*, 14(2):83–97, 1999.

[BRV98] Laura Bright, Louiqa Raschid, and Maria-Esther Vidal. Optimization of wrappers and mediators for web accessible data sources (websources). In *First Workshop on Web Information and Data Management, WIDM'98, Bathesda, Maryland, USA, November 6, 1998*, pages 40–44. ACM, 1998.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[CD97] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997.

[CGL01] Andrea Calì, Giuseppe De Giacomo, and Maurizio Lenzerini. Models for information integration: Turning local-as-view into global-as-view. In *Proceedings of International Workshop on Foundations of Models for Information Integration (10th Workshop in the series Foundations of Models and Languages for Data and Objects)*, 2001.

[CGLV00] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. What is view-based query rewriting? In *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases, KRDB'00, Berlin, Germany, August 21, 2000*, volume 29 of *CEUR Workshop Proceedings*, pages 17–27. CEUR-WS.org, 2000.

[CGLV03] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query containment. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems PODS'03, June 9-12, 2003, San Diego, CA, USA*, pages 56–67. ACM Press, 2003.

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 190–200. IEEE Computer Society, 1995.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing, STOC'1999, 2-4 May 1977, Boulder, Colorado, USA*, pages 77–90. ACM, 1977.

[Dar98] Adnan Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research (JAIR)*, 8:165–222, 1998.

[Dar00] Adnan Darwiche. On the tractable counting of theory models and its application to belief revision and truth maintenance. *The Computing Research Repository, CoRR*, cs.AI/0003044, 2000.

[Dar01] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

[Dar02] Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI'2002, and Fourteenth Conference on Innovative Applications of Artificial Intelligence, IAAI'2002. Edmonton, Alberta, Canada, July 28 - August 1, 2002*, pages 627–634. AAAI Press, 2002.

[Dar04] Adnan Darwiche. New advances in compiling cnf into decomposable negation normal form. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004. Valencia, Spain, August 22-27, 2004.*, pages 328–332. IOS Press, 2004.

[DG97a] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems PODS'97, Tucson, Arizona, May 12-14, 1997*, pages 109–116. ACM Press, 1997.

[DG97b] Oliver M. Duschka and Michael R. Genesereth. Query planning in infomaster. In *Proceedings of the 1997 ACM symposium on Applied computing SAC '97*, pages 109–111, New York, NY, USA, 1997. ACM Press.

[EMCGP99] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[FLM98] Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(3):59–74, 1998.

[FMR+99] Michael Franklin, George Mihaila, L. Raschid, Tolga Urhan, María Esther Vidal, and Vladimir Zadorozhny. Searching and querying wide-area distributed collections. In *Proceedings of Russian National Conference on Digital Libraries (invited paper)*, 1999.

[GMPQ+97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The tsimmis approach to mediation: Data models and languages. *Journal of Intelligent Information Systems JIIS*, 8(2):117–132, 1997.

[GRVB98] Jean-Robert Gruser, Louiqa Raschid, Maria-Esther Vidal, and Laura Bright. Wrapper generation for web accessible data sources. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, New York City, New York, USA, August 20-22, 1998, Sponsored by IFCIS, The Intn'l Foundation on Cooperative Information Systems*, pages 14–23. IEEE Computer Society, 1998.

[GT99] Fausto Giunchiglia and Paolo Traverso. Planning as model checking. In *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Proceedings. Durham, UK, September 8-10, 2005*, volume 1809 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1999.

[Hal00] Alon Y. Halevy. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):40–47, 2000.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001.

[Her01] Marc Herbstritt. zchaff: Modifications and extensions, 2001.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'2002, June 3-5, Madison, Wisconsin, USA*, pages 233–246. ACM, 2002.

[LMS95] Alon Y. Levy, Alberto O. Mendelzon, and Yehoshua Sagiv. Answering queries using views (extended abstract). In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS'1995, May 22-25, 1995, San Jose, California, USA.*, pages 95–104, New York, NY, USA, 1995. ACM Press.

[LRO96a] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI'1996, and Eighth Conference on Innovative Applications of Artificial Intelligence, IAAI'1996, August 4-8, 1996, Portland, Oregon*, pages 40–47. AAAI Press/ MIT Press, 1996.

[LRO96b] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases, VLDB'1996, September 3-6, 1996, Mumbai (Bombay), India*, pages 251–262. VLDB Endowment, 1996.

[LRO96c] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. The world wide web as a collection of views: Query processing in the information manifold. In *Proceedings of Workshop on Materialized Views: Techniques and Applications, VIEWS'96. Monteal, Canada, Friday, June 7, 1996, in cooperation with SIGMOD Conference 1996*, pages 43–55, 1996.

[LRU96] Alon Y. Levy, Anand Rajaraman, and Jeffrey D. Ullman. Answering queries using limited external processors. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, pages 227–237. ACM Press, 1996.

[Mit99] P. Mitra. An algorithm for answering queries efficiently using views. Technical report, Stanford University, Stanford, CA, USA, 1999.

[MLF00] Todd D. Millstein, Alon Y. Levy, and Marc Friedman. Query containment for data integration systems. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA.*, pages 67–75, 2000.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.

[PBDG05] Héctor Palacios, Blai Bonet, Adnan Darwiche, and Hector Geffner. Pruning conformant plans by counting models on compiled d-dnnf representations. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling, ICAPS'2005. Monterey, California, USA, June 5-10, 2005*, pages 141–150. AAAI Press, 2005.

[PH01] Rachel Pottinger and Alon Halevy. MiniCon: A scalable algorithm for answering queries using views. *VLDB Journal: Very Large Data Bases*, 10(2–3):182–198, 2001.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[Qia96] Xiaolei Qian. Query folding. In *Proceedings of the Twelfth International Conference on Data Engineering, New Orleans, Louisiana, February 26 - March 1, 1996*, pages 48–55. IEEE Computer Society, 1996.

[SS96] João P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *International Conference on Computer-Aided Design, ICCAD '96, November 10-14, 1996, San Jose, CA, USA.*, pages 220–227. ACM and IEEE Computer Society, 1996.

[TRV96] Anthony Tomasic, Louiqa Raschid, and Patrick Valduriez. Scaling heterogeneous databases and the design of disco. In *Proceedings of the 16th International Conference on Distributed Computing Systems ICDCS'96. May 27-30, 1996, Hong Kong*, pages 449–457, 1996.

[TRV98] Anthony Tomasic, Louiqa Raschid, and Patrick Valduriez. Scaling access to heterogeneous data sources with disco. *IEEE Transactions on Knowledge and Data Engeneering*, 10(5):808–823, 1998.

[TS97] Dimitri Theodoratos and Timos K. Sellis. Data warehouse configuration. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 126–135. Morgan Kaufmann, 1997.

[TSI96] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal: Very Large Data Bases*, 5(2):101–118, 1996.

[Ull00] Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science Journal*, 239(2):189–210, 2000.

[Vid00] M. E. Vidal. *A Mediator for Scaling up to Multiple Autonomous Distributed Information Sources*. PhD thesis, Universidad Simón Bolívar, 2000.

[VR98] María Esther Vidal and Louiqa Raschid. Websrcmed: A mediator for scaling up to multiple web accessible sources. Technical report, UMIACS, University of Maryland, 1998.

[VRG98] Maria-Esther Vidal, Louiqa Raschid, and Jean-Robert Gruser. A meta-wrapper for scaling up to multiple autonomous distributed information sources. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, New York City, New York, USA, August 20-22, 1998, Sponsored by IFCIS, The Intn'l Foundation on Cooperative Information Systems*, pages 148–157. IEEE Computer Society, 1998.

[Wie97] Gio Wiederhold. Mediators in the architecture of future information systems. In *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[YL87] H. Z. Yang and Per-Åke Larson. Query transformation for psj-queries. In *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, pages 245–254. Morgan Kaufmann, 1987.

[ZBR+00] Vladimir Zadorozhny, Laura Bright, Louiqa Raschid, Tolga Urhan, and Maria-Esther Vidal. Web query optimizer. pages 661–663. IEEE Computer Society, 2000.

[ZCL+00] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, and Monica Urata. Answering complex sql queries using automatic summary tables. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 105–116. ACM, 2000.

[ZRV+02] Vladimir Zadorozhny, Louiqa Raschid, Maria-Esther Vidal, Tolga Urhan, and Laura Bright. Efficient evaluation of queries in a mediator for websources. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*, pages 85–96. ACM, 2002.

Apéndice A

Descripción de la teoría para la representación de *MCDs*

En este capítulo se presenta la definición de las cláusulas de la teoría para generación de *MCDs*.

En la sección A.1, se recuerdan algunas definiciones necesarias para comprender la construcción de la teoría. En las secciones A.2 y A.3, se describen las variables y cláusulas de la teoría *MCD*, respectivamente. Finalmente, en la sección A.4, se presentan las cláusulas de minimización.

A.1. Definiciones preliminares

En primer lugar, recuérdese la definición del problema de encontrar las reescrituras de una consulta Q sobre un conjunto de vistas $\mathcal{V}$.

Dado una consulta conjuntiva $Q(\vec{X}) : -r_0(\vec{X}_0), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)$ y un conjunto de vistas $\mathcal{V} = V_0, V_1, \ldots, V_n$ donde cada $V_i = p_{0_i}(\vec{Y}_{0_i}), p_{1_i}(\vec{Y}_{1_i}), \ldots, p_{m_i}(\vec{Y}_{m_i})$. Una reescritura de Q usando las vistas $\mathcal{V}$ es una consulta conjuntiva cuyos predicados son $V_0, V_1, \ldots, V_n$.

Además, recuérdese el *MCD* simplificado presentado en la sección 5.1:

- La vista V usada en el *MCD*.
- El conjunto G de subobjetivos de Q que es cubierto por el *MCD*.
- Una relación τ : *Vars(Q)* $\times$ *Vars(V)*, en la cual un par (q, v) representa que la variable q de Q se cubre con la variable v de V. Se permite que existan variables de Q que correspondan a

más de una variable en V, cumpliéndose la siguiente relación entre τ, φ y h:

$$(q, v) \in \tau \Leftrightarrow \varphi(q) = h(v)$$

A.2. Variables de la teoría

- v_i indica que el *MCD* usa la vista i.
- g_j indica que el *MCD* cubre al subobjetivo j de la consulta.
- $z_{j,k,i}$ indica que el subobjetivo j de la consulta es cubierto por el subobjetivo k de la vista i.
- $t_{a,b}$ indica que la variable a de Q se corresponde con la variable b de la vista en la relación τ.

NOTAS:

1. Cuando $i = -1$ indica que el *MCD* es vacío, no corresponde a ninguna vista.
2. Las variables z son auxiliares para indicar que todas las variables de un subobjetivo de la consulta corresponden a un subobjetivo de la vista. Ésto se garantiza en las clausulas.

A.3. Cláusulas de la teoría

Cl 1. Por lo menos una vista debe usarse en el MCD.

$$\bigvee_{\{i:-1 \leq i \leq n\}} v_i$$

Cl 2. Se debe usar a lo sumo una vista en el MCD.

$$v_i \Rightarrow \neg v_j$$

para $\{i, j : 0 \leq i < j \leq n\}$.

Cl 3. Si el *MCD* no usa ninguna vista, entonces es vacío.

$$v_{-1} \Rightarrow \bigwedge_{\{j:0\leq j<m\}} \neg g_j$$

Cl 4. Si un *MCD* usa la vista V_i entonces debe cubrir algún subobjetivo de la consulta.

$$v_i \Rightarrow \bigvee_{\{j:0\leq j\leq m\}} g_j$$

para $\{i : 0 \leq i \leq n\}$.

Cl 5. Si ningún subobjetivo de la vista V_i cubre el subobjetivo j de la consulta entonces el *MCD* no puede cubrirlo.

$$v_i \Rightarrow \neg g_j$$

para $\{i, j : 0 \leq i \leq n \wedge 0 \leq j \leq m \wedge \neg(\exists k : 0 \leq k \leq m_i : r_j = p_{k_i})\}$.

Cl 6. Si en la relación τ se tiene $\{x_l \rightarrow y_b\}$, entonces debe usarse alguna vista V_i a la cual pertenezca la variable y_b.

$$t_{a,b} \Rightarrow \bigvee_{\{i:0\leq i\leq n \wedge x_b\in Vars(V_i)\}} v_i$$

para $a, b : x_a \in Vars(Q)$

Cl 7. Si se cubre el subobjetivo j de la consulta con el subobjetivo k de la vista V_i entonces, en la relación τ, todas las variables $\vec{X}_j$ se corresponden a las variables $\vec{Y}_{k_i}$

$$v_i \wedge z_{j,k,i} \Rightarrow \bigwedge_{\{a,b:\, x_a\in\vec{X}_j \wedge y_b\in\vec{Y}_{k_i}\}} t_{a,b}$$

para $\{i, j : 0 \leq i \leq n \wedge 0 \leq j \leq m\}$.

Cl 8. Si en la relación τ se tiene $\{x_a \rightarrow y_b\}$, entonces debe existir un subobjetivo j en la consulta que pueda cubrirse con un subobjetivo k de la vista V_i tal que $x_a \in \vec{X}_j, y_b \in \vec{Y}_{i_k}$.

$$v_i \wedge t_{a,b} \Rightarrow \bigvee_{\{j,k:x_a\in\vec{X}_j, y_b\in\vec{Y}_{i,k}\}} z_{j,k,i}$$

para $\{i, a, b : 0 \leq i \leq n \wedge x_a \in Vars(Q) \wedge y_b \in Vars(V_i)\}$.

Cl 9. Si un *MCD* usa la vista V_i y cubre el subobjetivo j de la consulta, entonces este subobjetivo

de la consulta debe ser cubierto por algún subobjetivo de la vista V_i.

$$v_i \wedge g_j \Leftrightarrow \bigvee_{\{k:p_{i,k}=r_j\}} z_{j,k,i}$$

para $\{i, j : 0 \leq i \leq n \wedge 0 \leq j \leq m\}$.

Cl 10. (*Restricción de homomorfismo de cabeza*) Si en la relación τ se tiene: $\{x_a \rightarrow y_b\}$ y y_b es existencial en la vista V_i, entonces la variable x_a no puede cubrirse con ninguna otra variable.

$$v_i \wedge t_{a,b} \Rightarrow \neg t_{a,c}$$

para $\{i : 0 \leq i \leq n \wedge x_a \in Vars(Q) \wedge y_b \in Exist(V_i) \wedge x_a \neq y_b\}$.

Cl 11. (*Propiedad 1 C1*) En la relación τ, una variable distinguible en la consulta no puede cubrirse con una variable existencial de una vista.

$$v_i \Rightarrow \neg t_{a,b}$$

para $\{i, a, b : 0 \leq i \leq n \wedge x_a \in Dist(Q) \wedge y_b \in Exist(V_i)\}$.

Cl 12. (*Propiedad 1 C2*) Si en la relación τ se tiene: $\{x_a \rightarrow y_b\}$ y y_b es existencial en la vista V_i, entonces deben cubrirse todos los subobjetivos de la consulta en los que se encuentre la variable x_a.

$$v_i \wedge t_{a,b} \Rightarrow \bigwedge_{x_a \in X_j} g_j$$

para $\{i : 0 \leq i \leq n \wedge y_b \in Exist(V_i)\}$.

Cl 13. En un MCD, un subobjetivo de la consulta es cubierto por a lo sumo un subobjetivo de la vista.

$$z_{j,k,i} \Rightarrow \neg z_{j,l,i}$$

para $\{i, j, k, l : 0 \leq i \leq n \wedge 0 \leq j \leq m \wedge 0 \leq k < l \leq m_i \wedge r_j = p_{k_i} = p_{l_i}\}$.

A.4. Cláusulas de minimización

Las cláusulas que se presentan a continuación, no son necesarias para la correctitud de la teoría. Sin embargo, en ciertas situaciones, las siguientes cláusulas puede reducir el número de *MCDs* que se producen.

Cl 14. Si todas las variables de la vista V_i son distinguibles entonces el *MCD* a lo sumo cubre un subobjetivo de la consulta.

$$v_i \wedge g_j \Rightarrow \neg g_k$$

para $\{i, j, k : 0 \leq i < j \leq n \wedge 0 \leq i \leq n \wedge V_i$ tiene todas las variables distinguibles $\}$.

Apéndice B

Demostración del teorema 5.2

En este capítulo se presenta la demostración del teorema 5.2.

En la sección B.1, se recuerdan algunas definiciones necesarias para realizar la demostración y se enuncia el teorema nuevamente. En las secciones B.2 y B.3, se presentan las demostraciones de correctitud y completitud del teorema, respectivamente.

B.1. Definiciones preliminares

En primer lugar, recuérdese la definición del problema de encontrar las reescrituras de una consulta Q sobre un conjunto de vistas $\mathcal{V}$.

Dado una consulta conjuntiva $Q(\vec{X}) : -r_0(\vec{X}_0), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)$ y un conjunto de vistas $\mathcal{V} = V_0, V_1, \ldots, V_n$ donde cada $V_i = p_{0_i}(\vec{Y}_{0_i}), p_{1_i}(\vec{Y}_{1_i}), \ldots, p_{m_i}(\vec{Y}_{m_i})$. Una reescritura de Q usando las vistas $\mathcal{V}$ es una consulta conjuntiva cuyos predicados son $V_0, V_1, \ldots, V_n$.

Además, recuérdese el *MCD* simplificado presentado en la sección 5.1:

- La vista V usada en el *MCD*.
- El conjunto G de subobjetivos de Q que es cubierto por el *MCD*.
- Una relación τ : *Vars(Q)* $\times$ *Vars(V)*, en la cual un par (q, v) representa que la variable q de Q se cubre con la variable v de V. Se permite que existan variables de Q que correspondan a más de una variable en V, cumpliéndose la siguiente relación entre τ, φ y h:

$$(q, v) \in \tau \Leftrightarrow \varphi(q) = h(v)$$

A continuación se enuncia nuevamente el teorema 5.2:

Teorema (5.2). *Sea C un MCD para el problema de reescribir la consulta Q usando las vistas $\mathcal{V}$. Sea T la teoría generada para el problema. Si C es minimal, entonces existe un modelo ω para T tal que $C = C_\omega$. Si ω es un modelo para T entonces ω es vacío o M_ω es un MCD.*

B.2. Prueba de correctitud

Para demostrar la correctitud del teorema, se necesita mostrar que todo modelo de la teoría es un *MCD* para el problema de reescrituras o es el *MCD* vacío.

Sea ω un modelo para la teoría construida como se indica en el apéndice A para el problema de reescribir una consulta Q usando las vistas $\mathcal{V}$. ω es el *MCD* vacío, si la variable $v_{-1} = True$. Si no, entonces es posible construir la tupla $C = (V_C, G_C, \tau_C)$, a partir de ω, de la siguiente manera:

- $V_c = v_i$ para la variable $v_i = True$. Debido a la definición de las clausulas **Cl 1**, **Cl 2** se garantiza que existe exactamente una variable $v_i = True$ para $0 \leq i \leq m$.
- $j \in G_C$ si y solo si $g_j = True$ en ω. Por la definición de las cláusulas **Cl 4**, **Cl 5** se garantiza que $G_C \neq \emptyset$ y que solo se cubren subobjetivos de la consulta con subobjetivos consistentes de la vista.
- $(a, b) \in \tau$ si y solo si $t_{a,b} = True$ en ω. Para ver que τ es válido, nótese los siguiente:

  En primer lugar, τ cubre las variables de la consulta con variables de la vista V_C debido a la definición de las cláusulas **Cl 6**. Además, al cubrir un subobjetivo de la consulta, deben cubrirse todas las variables que se encuentren en el subobjetivo, por la definición de las cláusulas **Cl 7** y **Cl 8**. Y por la definición de las cláusulas **Cl 9** se garantiza, que τ cubre a todos los subobjetivos que se encuentran en G_C.

Finalmente, para demostrar que la tupla C, construida de esta manera, corresponde a un *MCD* válido para el problema de reescrituras, es necesario demostrar que es posible transformar τ_C en φ_C y h_C válidos.

Por la definición de las cláusulas **Cl 10**, se tiene que τ cumple la propiedad de homomorfismo de cabeza, en la cual, sólo se permite igualar variables distinguibles de la vista.

Finalmente, por la definición de las cláusulas **Cl 11**, **Cl 12**, se tiene que τ_C cumple la propiedad 1, con lo cual se concluye que C es un *MCD* válido.

B.3. Prueba de completitud

Se necesita mostrar que todo *MCD* minimal para el problema de reescrituras es un modelo en la teoría. Para ello se construirá una asignación de variables para la teoría lógica usando la información contenida en el *MCD* y luego se mostrará que esta asignación satisface todas las cláusulas de la teoría.

Sea C un *MCD* $(h_C, V_C, \varphi_C, G_C)$ para el problema de reescrituras de la consulta:

$$Q(\vec{X}) : -r_0(\vec{X}_0), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)$$

usando el conjunto de vistas $\mathcal{V} = V_0, V_1, \ldots, V_n$. Donde, $V_i = p_{0_i}(\vec{Y}_{0_i}), p_{1_i}(\vec{Y}_{1_i}), \ldots, p_{m_i}(\vec{Y}_{m_i})$.

A partir de φ_C y h_c, se define la relación τ:

$$\tau = \{(x, y) : y \in Vars(V_c) \ \wedge \ \varphi_c(x) = h_C(y)\}$$

La asignación de variables ω se construye, a partir de C, de la siguiente manera.

P.1 $v_i = \texttt{True}$, si y solo $V_c = v_i$. $v_{-1} = \texttt{False}$.

P.2 $g_j = \texttt{True}$, si y solo si $i \in G_C$.

P.3 $z_{j,k,i} = \texttt{True}$, si el *MCD* cubre el subobjetivo j de la consulta con el subobjetivo k de la vista i, es decir, si se cumple que $\varphi(r_j(\vec{X}_j)) = h(p_{k_i}(\vec{Y}_{k_i}))$.

P.4 $t_{a,b} = \texttt{True}$, indica que la variable a de Q se cubre con la variable b de la vista en la relación τ.

Ahora se debe mostrar que toda asignación de variables ω, construida de esta manera, es un modelo de la teoría. Es decir, falta demostrar que ω satisface todas las cláusulas de la teoría, descritas en el apéndice A.

Las cláusulas **Cl 1**, **Cl 2** y **Cl 3** se satisfacen por la asignación de variables ω (**P. 1**) ya que el *MCD* contiene exactamene una vista.

La cláusula **Cl 4** se satisface debido a que si el *MCD* es no vacío entonces $G_C \neq \emptyset$.

Las cláusulas **Cl 5** se satisfacen debido a que el *MCD M* solo cubre la consulta con subobjetivos consistentes de la vista.

Las cláusulas **Cl 6** se satisfacen debido a que si se tiene $\varphi_C(x) = h_C(y)$ entonces la variable y debe pertenecer a la vista.

Si en el *MCD* se usa el subobjetivo k de la vista i para cubrir el subobjetivo j de la consulta, se cumple que $\varphi_C(r_j(\vec{X}_j)) = h_C(p_{k_i}(\vec{Y}_{k_i}))$. Luego, por la construcción de τ se deben tener todos los pares (x, y) tales que $x_l \in \vec{X}_j$ y $y_l \in \vec{Y}_{k_i}$, con lo cual se satisfacen las cláusulas **Cl 7** y **Cl 8**.

Por la definición de *MCD*, debe existir en V_C un subobjetivo que cubra a cada uno de los subobjetivos de Q que se encuentran en G_C, por lo cual las cláusulas **Cl 9** son satisfechas.

Para mostrar que se satisfacen las cláusulas **Cl 10**, **Cl 11** y **Cl 12**, se utilizará reducción al absurdo.

Supongamos que no se cumple la cláusula: $v_i \wedge t_{a,b} \Rightarrow \neg t_{a,c}$ donde $y_b \in Exist(V_C)$. Entonces, se tiene que existe $v_i = V_C$ y $(x_a, y_b), (x_a, y_c) \in \tau$ y por ende $h_C(y_b) = h_C(y_c)$. Pero esto es una contradicción, debido a que por la definición de homomorfismo de cabeza, h_C solo puede igualar variables distinguibles en la cabeza de la vista, por lo cual se concluye, que las cláusulas **Cl 10** son satisfechas por ω.

Por otro lado, supongamos que no se cumple la cláusula: $v_i \Rightarrow \neg t_{a,b}$ donde $x_a \in Dist(Q)$, $y_b \in Exist(V_i)$. Entonces, se tiene que $v_i = V_C$ y $(x_a, y_b) \in \tau$ lo cual es una contradicción, dado que por la definición de propiedad 1, una variable distinguible en Q no puede cubrirse con una variable existencial en V_C. Con lo cual se concluye que las cláusulas **Cl 11** son satisfechas.

Ahora, supongamos que no se cumple la cláusula: $v_i \wedge t_{a,b} \Rightarrow g_j$ donde $y_b \in Exist(V_i)$ y $x_a \in X_j$. Entonces, se tiene que si $v_i = V_C$ y $(x_a, y_b) \in \tau$ entonces $g_j \notin G_C$. Lo cual es una contradicción, dado que por la definición de *MCD* y propiedad 1, si se tiene que la variable y_b es existencial en V_C entonces todos los *join* donde participe x_a son cubiertos con V_c. Por lo tanto las cláusulas **Cl 12** son satisfechas.

Por último, falta demostrar las cláusulas de minimización. Las cláusulas **Cl 14** se satisfacen en el caso de que todas las variables de la vista sean distinguibles. Por reducción al absurdo supongamos que no se cumple la cláusula $v_i \wedge g_j \Rightarrow \neg g_k$ donde V_i tiene todas las variables distinguibles. Luego, tenemos que $g_j, g_k \in G_C$. Pero, dado que g_j tiene todas las variables distinguibles entonces, en el *mapping* τ, g_j puede cubrir por si solo cualquier subobjetivo de Q, que sea consistente con g_j. Análogamente, g_k puede cubrir por si solo cualquier subobjetivo de Q consistente usando el

mapping τ. Por lo tanto el *MCD C* no es minimal, de lo cual se obtiene una contradicción que concluye que las cláusulas **Cl 14** son satisfechas.

Finalmente, se concluye que todas las cláusulas de la teoría son satisfechas por ω, por lo tanto, el *MCD C* es un modelo de la teoría.

Apéndice C

Descripción de la teoría extendida

En este capítulo se presenta la definición de las cláusulas de la teoría para generación de reescrituras.

En las secciones C.1 y C.2, se describen las variables y cláusulas de la teoría extendida.

C.1. Variables de la teoría extendida

Al crear las copias de la teoría *MCD*, se agrega a las variables un superíndice t para indicar que la misma pertenece a la $t-$ésima copia. Es decir, las variables de la teoría extendida son de la forma:

- v_i^t
- g_j^t
- $z_{j,k,i}^t$
- $t_{a,b}^t$

conservando el mismo significado que tenían en la teoría *MCD*, descrita en el apéndice A.

C.2. Cláusulas de la teoría extendida

Las cláusulas de la teoría extendida están formadas por m copias de las cláusulas de la teoría *MCD*, agregando cláusulas que garantizan que cada subobjetivo es cubierto exactamente una vez y que no se producen reescrituras simétricas.

A continuación se describen las cláusulas adicionales para la teoría extendida:

Cl 15. Todos los subobjetivos de Q deben ser cubiertos por el *MCD* de alguna copia.

$$\bigvee_{0 \leq t < n} g_j^t$$

para $\{j : 0 \leq j < m\}$.

Cl 16. Los *MCDs* cubren subobjetivos disjuntos de Q.

$$g_j^t \Rightarrow \neg g_j^s$$

para $\{j, s, t : 0 \leq j, s, t < m \wedge s < t\}$.

Cl 17. Si el *MCD* de la copia t cubre el subobjetivo j de la consulta, entonces el *MCD* de la copia $t-1$ debe cubrir un subobjetivo k con $0 \leq k < j$.

$$g_j^t \Rightarrow \bigvee_{0 \leq k < j} g_j^{t-1}$$

para $\{i, t : 0 \leq i, t < m\}$.

Apéndice D

Demostración del teorema 5.3

En este capítulo se presenta la demostración del teorema 5.3.

En la sección D.1, se recuerdan algunas definiciones necesarias para realizar la demostración y se enuncia el teorema nuevamente. En las secciones D.2 y D.3, se presentan las demostraciones de correctitud y completitud del teorema para teorías extendida, respectivamente.

D.1. Definiciones preliminares

En primer lugar, recuérdese la definición del problema de encontrar las reescrituras de una consulta Q sobre un conjunto de vistas $\mathcal{V}$.

Dado una consulta conjuntiva $Q(\vec{X}) : -r_0(\vec{X}_0), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)$ y un conjunto de vistas $\mathcal{V} = V_0, V_1, \ldots, V_n$ donde cada $V_i = p_{0_i}(\vec{Y}_{0_i}), p_{1_i}(\vec{Y}_{1_i}), \ldots, p_{m_i}(\vec{Y}_{m_i})$. Una reescritura de Q usando las vistas $\mathcal{V}$ es una consulta conjuntiva cuyos predicados son $V_0, V_1, \ldots, V_n$.

A continuación se enuncia nuevamente el teorema 5.3:

Teorema (5.3). *Sea T_E la teoría extendida generada a partir de Q y $\mathcal{V}$. ω_E es un modelo para T_E si y solo si el conjunto $\mathcal{C}_{\omega_E}$ de MCDs obtenidos a partir de ω_E forman una reescritura válida para Q.*

D.2. Prueba de correctitud

Para demostrar la correctitud del teorema, se necesita mostrar que todo modelo de la teoría es una reescritura para el problema.

Sea ω_E un modelo para la teoría construida como se indica en el apéndice C para el problema de reescribir una consulta Q usando las vistas $\mathcal{V}$.

Utilizando el superíndice t y el teorema 5.2, se pueden recuperar los *MCD* C_t correspondientes a cada una de las copias de la teoría *MCD*.

Falta demostrar que el conjunto de *MCDs* $S = \{C_t : 0 \leq t < m\}$ constituye una reescritura válida para el problema. Por la definición de las cláusulas **Cl 15** y **Cl 16** se tiene que los *MCDs* en S cubren todos los subobjetivos de Q. Luego, por la propiedad 2, el resultado de combinar los *MCDs* en S es un reescritura válida.

Finalmente, se concluye que ω_E corresponde a una reescritura válida.

D.3. Prueba de completitud

Para probar completitud, se necesita mostrar para toda reescritura del problema existe un modelo en la teoría. Para ello se construirá una asignación de variables para la teoría lógica a partir de la reescritura y luego se mostrará que esta asignación satisface todas las cláusulas de la teoría extendida.

Sea Q' una reescritura para el problema de reescrituras de la consulta:

$$Q(\vec{X}) : -r_0(\vec{X}_0), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)$$

usando el conjunto de vistas $\mathcal{V} = V_0, V_1, \ldots, V_n$. Donde, $V_i = p_{0_i}(\vec{Y}_{0_i}), p_{1_i}(\vec{Y}_{1_i}), \ldots, p_{m_i}(\vec{Y}_{m_i})$.

Dado que Q' es una reescritura para Q entonces existe un *containment mapping* θ_Q de Q a la expansión E' de Q', tal que $\theta_Q(Q(\vec{X})) = Q'(\vec{X}')$, es decir, toda variable distinguible en Q se corresponde en una variable distinguible en Q'.

Sea G_i el conjunto de subobjetivos de Q que se cubren con algún subobjetivo de la vista V_i en el *mapping* θ_Q. Sea θ_t igual al *mapping* θ_Q restringido a las variables de los subobjetivos de G_t.

Por la demostración de completitud del algoritmo de *Minicon* [PH01], es posible transformar θ_t, G_t en el *MCD* C_t tal que satisface la propiedad 1. Además el conjunto de *MCDs* $\mathcal{C}$, formados de esta manera, satisfacen la propiedad 2.

Se ordenan los *MCDs* $C_t \in \mathcal{C}$, de la siguiente manera: $C_t < C_r$ si y solo si mín$(G_t) <$ mín(G_r). Siguiendo este orden, se construye la asignación de variables ω_E, para la teoría extendida, de la siguiente manera.

P.1 $v_i^t =$ `True`, si y solo $V_{c_t} = v_i$. Si $v_{-1} =$ `True` entonces el *MCD* C_t es vacío.

P.2 $g_j^t =$ `True`, si y solo si $i \in G_{C_t}$.

P.3 $z_{j,k,i}^t =$ `True`, si el *MCD* C_t cubre el subobjetivo j de la consulta con el subobjetivo k de la vista i, es decir, si se cumple que $\varphi_{C_t}(r_j(\vec{X}_j)) = h_{C_t}(p_{k_i}(\vec{Y}_{k_i}))$.

P.4 $t_{a,b}^t =$ `True`, indica que la variable a de Q se cubre con la variable b de la vista en la relación τ_{C_t}.

Ahora se debe mostrar que cualquier asignación de variables ω_E, construida de esta manera, es un modelo de la teoría. Para ello, se debe demostrar que ω_E satisface todas las cláusulas de la teoría, descritas en el apéndice A.

Por el teorema 5.2, las cláusulas de cada copia de la teoría *MCD* se satisfacen. Falta demostrar que se satisfacen las cláusulas agregadas a la teoría extendida.

Las cláusulas **Cl 15** y **Cl 16** se satisfacen debido a que los *MCDs* en $\mathcal{C}$ cumplen la propiedad 2. Finalmente, las cláusulas **Cl 17** se cumplen por el orden impuesto a los *MCDs* al crear ω_E.

Apéndice E

Manual MCDSAT

En este capítulo se presenta el funcionamiento de MCDSAT. La sección E.1, describe el funcionamiento de cada uno de los componentes de MCDSAT. La sección E.2, enumera los requisitos de instalación y finalmente en la sección E.3 se describe la forma de ejecutar la herramienta.

E.1. Componentes

En esta sección se describen cada uno de los componentes de MCDSAT.

E.1.1. Generación de la teoría lógica

El primer componente de MCDSAT traduce el problema de reescrituras en la teoría MCD/Extendida correspondiente usando las definiciones presentadas en los anexos A y C. Para esto se implementó un generador, en el lenguaje Python, que produce las cláusulas en CNF.

Los parámetros que recibe el generador son los siguientes:

- Tipo de teoría a generar: MCD o extendida.
- Definición de la consulta a partir de un archivo de texto que sigue el formato para especificar consultas conjuntivas, que se presenta en la sección E.3.1.
- Definición del conjunto de vistas. Se recibe en un archivo de texto con el mismo formato que la consulta.

Una vez ejecutado, el generador produce:

- La teoría equivalente al problema, bien sea teoría MCD o extendida. La teoría se genera en el formato DIMACS [1]. Este formato genera las variables proposicionales como una secuencia de números.
- Un archivo de texto que contiene la equivalencia entre las variables proposicionales y los componentes del problema de reescrituras. Este archivo, es usado posteriormente por el generador de *MCDs*/Reescrituras.
- Un archivo de texto que contiene el tiempo de generación de las cláusulas.

E.1.2. Compilación a DNNF

Se utilizó el compilador `c2d` [Dar04] para transformar la teoría CNF a d-DNNF. Este compilador recibe la teoría CNF especificada en el formato DIMACS y produce la teoría compilada a d-DNNF. Para una descripción completa del funcionamiento de este compilador y del formato de especificación del d-DNNF que produce ver `http://reasoning.cs.ucla.edu/c2d/`.

MCDSAT utiliza los siguientes parámetros del compilador:

- El *dtree* se construye usando el orden de inverso de las variables.
- Se fuerza la propiedad de *Smoothness* del d-DNNF generado.
- Los demás parámetros son los definidos por defecto.

E.1.3. Obtención de modelos a partir del DNNF

Una vez generado el d-DNNF, los modelos se obtienen utilizando la herramienta MODELS, desarrollada por el profesor Blai Bonet.

MODELS recibe como entrada el d-DNNF producido por `c2d` y genera una lista de modelos. Donde cada modelo se representa con los nombres de las variables p cuya asignación fue $p =$ `True`. Si la variable p no aparece, se asume que su asignación fue $p =$ `False`.

[1] Una descripción de este formato se encuentra disponible en `www.satlib.org/Benchmarks/SAT/satformat.ps`

E.1.4. Generación de *MCDs*/Reescrituras

El componente final de MCDSAT transforma los modelos obtenidos por MODELS, en *MCDs*/reescrituras del problema. Para ello se implementó un programa en Python que realiza las transformaciones descritas en las definiciones 5.1 y 5.2.

Los parámetros que recibe el generador son los siguientes:

- Tipo de teoría: MCD o extendida.
- Definición de la consulta a partir de un archivo de texto que sigue el formato para especificar consultas conjuntivas, que se presenta en la sección E.3.1.
- Definición del conjunto de vistas. Se recibe en un archivo de texto con el mismo formato que la consulta.
- El conjunto de modelos producidos por MODELS.
- El archivo de texto que contiene la equivalencia entre las variables proposicionales y los componentes del problema de reescrituras.

Una vez ejecutado, el generador produce:

- El conjunto de *MCDs*/Reescrituras equivalentes a los modelos que recibió como parámetro.
- Un archivo de texto que contiene el tiempo de transformación de los modelos.

E.2. Requerimientos

- Para los componentes implementados en Python se requiere la versión 2.4 o superior. Además, se requieren las librerías Psyco y Tpg.
- Para MODELS, se requiere el compilador de C++ y la librería GMP.
- Para los requerimientos de instalación del compilador ver `http://reasoning.cs.ucla.edu/c2d/`.

E.3. Ejecución de MCDSAT

La manera mas sencilla de ejecutar MCDSAT es la siguiente:

```
$> mcdsat.sh <tipo_teoria> <arch_vistas> <arch_consultas>
```

donde:

- `<tipo_teoria>`: Puede tomar dos valores: `MCD` para generar la teoría MCD y `RW` para generar la teoría extendida.
- `<arch_vistas>`: Es un archivo que contiene la definición de las vistas del problema. Para ver el formato en que se definen consultas conjuntivas ver la sección E.3.1.
- `<arch_consultas>`: Es un archivo que contiene la definición de la consulta del problema. Para ver el formato en que se definen consultas conjuntivas ver la sección E.3.1.

También es posible ejecutar cada componente de MCDSAT por separado. Mas aun, es posible cambiar el compilador, el lenguaje de compilación o el generador de modelos siempre y cuando el nuevo componente sea compatible con el formato DIMACS y produzca los modelos de la manera que se describió en la sección E.1.3.

E.3.1. Sintaxis del archivo de consultas conjuntivas

Un archivo de consultas conjuntivas se usa para definir las vistas o consultas de un problema de reescrituras. Cada archivo debe contener por lo menos una consulta conjuntiva, pero puede contener más, si el archivo corresponde a la definición de un conjunto de vistas.

La sintaxis para la definición de una consulta conjuntiva se presenta a continuación:

```
CC ::= SO ":-" CUERPO

CUERPO ::= SO *("," SO)

SO ::= nomb LISTA_VAR

LISTA_VAR ::= "(" var  *("," var)  ")"
```

```
var ::= [A-Z][A-Za-z0-9]*

nomb ::= [a-z][a-z0-9]*
```

El siguiente ejemplo presenta el contenido de un archivo para la definición de las vistas del ejemplo 3.3:

```
v1(X1,X2) :- flight(X1,X2), uscity(X1), uscity(X2)
v2(X1,X2) :- flight(X1,X2)
v3(X1,X3) :- flight(X1,X2),flight(X2,X3)
```