UNIVERSIDAD DE COSTA RICA
SISTEMA DE ESTUDIOS DE POSGRADO

DESCUBRIMIENTO DE PATRONES SECUENCIALES UTILIZANDO RAZONAMIENTO LÓGICO TEMPORAL

Tesis sometida a la consideración de la Comisión del Programa de Estudios de Posgrado en Computación e Informática para optar al grado y título de Maestría Académica en Computación e Informática

BRAULIO JOSÉ SOLANO ROJAS

CIUDAD UNIVERSITARIA RODRIGO FACIO, COSTA RICA

2011

Dedicatoria Dedico esta tesis a mi familia: mi esposa, mis padres y mis hermanas.

Agradecimientos En primer lugar agradezco a mi director de tesis por su paciencia, por su perseverancia y por haberme aceptado como su tesiario. Quisiera agradecer, también, a mi comité asesor por sus consejos, su gran apoyo y por portarse como unos padres y una madre académicos. Agradezco, también, los clientes o las empresas que durante estos años de estudiante de maestría me brindaron oportunidades económicas. Por último, quisiera agradecer, de manera general, pues la lista sería larga, a todos aquellos que de alguna manera directa o indirecta fueron un apoyo: familiares, amigos, abogado, profesores, compañeros, colegas y otros.

“Esta tesis fue aceptada por la Comisión del Programa de Estudios de Posgrado en Computación e Informática de la Universidad de Costa Rica, como requisito parcial para optar al grado y título de Maestría Académica en Computación e Informática”

Dra. Gabriela Marín Raventós

Decana del Sistema de Estudios de Posgrado

Dr. José Ronald Argüello Venegas

Director de tesis

Dra. Elzbieta Malinowski

Asesora

Dr. Oldemar Rodríguez Rojas

Asesor

Dr. Vladimir Lara Villagrán

Director del Programa de Posgrado en Computación e Informática

Braulio José Solano Rojas

Candidato

Índice general

Resumen

Los datos secuenciales pueden ser recolectados en muchas aplicaciones como registros de ventas, bolsa de valores, registros médicos de pacientes, bases de datos en geofísica y astronomía, entre otras aplicaciones. Tales bases de datos incorporan la dimensión de tiempo que describe cuando ocurren eventos. La naturaleza temporal de los datos brinda un mejor entendimiento de las tendencias o patrones en el tiempo con el fin de encontrar relaciones entre eventos. Es posible combinar este hecho con bases teóricas que han sido desarrolladas por la filosofía, la matemática y las ciencias de la computación, tal como la lógica modal temporal. Sin embargo, los algoritmos existentes en descubrimiento de patrones secuenciales no utilizan el formalismo de la lógica temporal. Por ello, se desarrolla un método para descubrir relaciones temporales entre eventos en datos secuenciales utilizando la lógica temporal como base teórica.

Así, se describe un modelo de descubrimiento de patrones secuenciales en el cual se incorpora la lógica temporal modificando el algoritmo *Patrones Secuenciales Generalizados* (GSP, por sus siglas en inglés). Se implementa el modelo por medio de una combinación de componentes de código abierto, además de la programación del algoritmo que incorpora la lógica temporal. Se realiza un estudio de eficacia sobre un conjunto de datos artificial y los resultados muestran la eficacia del modelo propuesto.

Summary

Sequential data can be collected in many applications such as sales records, stock market, patient's medical records, databases in geophysics and astronomy, among other applications. Such databases incorporate the time dimension that describes when events occur. The temporal nature of data provides a better understanding of trends or patterns over time so as to find relationships between events. This can be combined with theoretical foundations that have been developed in philosophy, mathematics and computer science, such as temporal modal logic. A method was developed to discover temporal relationships between events in sequential data using temporal logic as a theoretical basis.

Existing algorithms for sequential pattern discovery are not using the formalism of temporal logic. Thus, a model of sequential patterns discovery is described in which temporal logic is incorporated by modifying the algorithm Generalized Sequential Patterns (GSP). The model is implemented through a combination of open source components as well as the programming of the algorithm that incorporates the temporal logic. An efficacy study on a set of artificial data was performed and the results show the effectiveness of the proposed model.

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

La investigación de esta tesis surge de la aplicación de la lógica temporal en la minería de datos para la búsqueda de patrones secuenciales. Este primer capítulo está divido en seis partes: minería de datos temporales, justificación académica, importancia social, objetivos y alcance, metodología y, finalmente, contenido y contribuciones.

1.1. Minería de datos temporal

La minería de datos ha pasado por el proceso de convertirse de hallazgo fortuito en una ciencia. Esta disciplina, ha tenido sus raíces ideológicas en la inteligencia artificial, la estadística, el análisis de datos y los algoritmos (Ramakrishnan & Grama, 1999). Esta ciencia o disciplina ha crecido y posee un carácter en sí misma. Este crecimiento se debe a la aparición de la Internet como gran repositorio de datos distribuidos y a la toma de conciencia de que las grandes bases de datos en línea pueden ser utilizadas para una ganancia comercial significante (Ramakrishnan & Grama, 1999).

Los objetivos comunes de las aplicaciones de minería de datos son la detección, la interpretación y la predicción de patrones, cuantitativos y cualitativos, en los datos, de manera automática. La minería de datos utiliza para caracterizar y evaluar patrones una variedad de modelos heredados del aprendizaje de máquinas, la estadística, la algorítmica experimental, la inteligencia artificial y las bases de datos. Estas disciplinas además vienen de enfoques matemáticos tales como la teoría de la aproximación y los sistemas dinámicos (Ramakrishnan & Grama, 1999).

Existen muchas técnicas de minería de datos y éstas se pueden clasificar desde bastantes perspectivas diferentes. Una de estas perspectivas es el dato en el que operan las técnicas: continuo, temporal, discreto, etiquetado o nominal.

Esta investigación profundizó en la técnica que puede operar sobre el dato que podemos clasificar como **temporal**, de manera específica las **secuencias**. Esto es importante dado que podemos encontrar no sólo secuencias, sino que también se puede encontrar (TASDA, 2002):

> "*Series de tiempo*: listas de valores numéricos muestreados uniformemente, las cuales se encuentran típicamente en sistemas técnicos.
>
> *Fragmentos de series de tiempo*: secuencias de series de tiempo, en donde cada serie es llamada un fragmento, con brechas intermedias. Los diferentes fragmentos pueden utilizar tasas de muestreo diferentes. Se encuentran típicamente en el dominio de las ciencias médicas.
>
> *Secuencias de eventos*: listas de eventos con marcas de tiempo.
>
> *Secuencias de estado o de intervalo*: listas de intervalos en el tiempo que denotan que los objetos permanecen en un cierto estado durante un tiempo, hasta que cierto evento ocurra y cambie el estado."

Así, un ejemplo de patrón secuencial es "*Un cliente que compra una computadora personal Pentium nueve meses antes probablemente comprará un chip de CPU nuevo en un mes*" (Han & Kamber, 2001). Otro ejemplo, de patrón sería "*El cliente renta* La guerra de las galaxias*, luego* El imperio contraataca *y luego* El regreso del Jedi".

Los datos temporales o series de tiempo requieren de un tratamiento especial en la minería de datos, siendo este tratamiento la base de esta investigación.

1.2. Justificación académica

En general, en la minería de datos clásica, o sea, la minería que no trata específicamente el análisis de datos secuenciales o temporales, el proceso del descubrimiento de conocimiento puede ser descompuesto en el siguiente ciclo (Han & Kamber, 2001):

> "*Limpieza de los datos*: para quitar el ruido y los datos inconsistentes.
> *Integración de los datos*: se pueden combinar varias fuentes de datos.
> *Selección de los datos*: se recuperan de la base de datos los datos relevantes a la tarea de análisis de los datos.
> *Transformación de los datos*: se transforman o consolidan los datos en formas apropiadas para minería, como por ejemplo realizando operaciones tales como agregación o suma.
> *Minería de datos*: es el proceso esencial donde se utilizan métodos inteligentes para extraer los patrones de datos.
> *Evaluación de los patrones*: se identifican los patrones verdaderamente interesantes, que representan conocimiento, utilizando alguna medida de interés.
> *Presentación del conocimiento*: se utilizan técnicas de visualización y de re-

presentación del conocimiento para presentar el conocimiento extraído al usuario."

Muchos enfoques utilizan los cuatro primeros pasos para adaptar y transformar los datos y así poder utilizar los algoritmos clásicos de minería de datos. Sin embargo, cabe la pregunta: ¿Es suficiente adaptar y transformar los datos temporales y aplicar algoritmos clásicos sobre ellos o es necesaria una minería de datos especializada para estos datos? La respuesta es que resulta necesario tener algoritmos especializados para los datos temporales, debido a que el análisis de datos temporales tiene características diferentes del análisis de datos no temporales (TASDA, 2002) como se describe a continuación.

1.2.1. Dilatación y traducción

Uno de los problemas con los datos temporales es, por ejemplo, cuando se transforman series de tiempo con la transformada de óndula o con la transformada de Fourier (técnicas de análisis de señales) para convertir datos del dominio de tiempo al dominio de frecuencia; el "ruido" en el sistema observado no conlleva "ruido" en las variables de salida, sino más bien a efectos de dilatación o de traducción. Es decir, dado que las transformadas traducen de un dominio a otro y producen una nueva escala, simplemente se está dilatando y traduciendo el ruido. Además, muchos de los algoritmos de minería de datos estáticos asumen propiedades como una distribución Gausiana en las variables observadas para manejar el "ruido". Esta suposición no se cumple para los datos temporales, para manejar el "ruido" se necesita información adicional sobre los valores de las variables inspeccionando su vecindario temporal (TASDA, 2002).

1.2.2. Similitud

La noción de similitud es utilizada cuando se comparan datos temporales. La similitud está muy relacionada a la característica anterior, debido a que la distancia euclidiana entre dos señales es igual en el dominio de tiempo y en el dominio de frecuencia (Han & Kamber, 2001).

Se puede ilustrar por medio de un ejemplo gráfico como los métodos clásicos para encontrar similitud no aplican cuando se tratan datos temporales. Nótese que en la figura 1.2.1, dado el hecho de que los procesos temporales raramente se repiten idénticamente, es muy probable que las curvas (a), (b) y (d) en la figura correspondan a situaciones similares en un sistema observado. Situaciones similares deberían ser emparejadas con patrones idénticos o similares del espacio de patrones. Sin embargo, si se consideran las curvas como vectores de valores de funciones y se aplica distancia euclidiana[1], tal como lo hacen muchos algoritmos de agrupación estática, la curva (a) se encuentra más cercana a (c) que a (b) o (d). Esto podría tener consecuencias graves para la interpretación de los resultados, i.e. si los patrones (b) o (d) son enseñados a humanos, seguramente su comprensión de la similitud va a incluir (a).

[1]Dadas dos series de tiempo $A = a_1, \ldots, a_n$ y $B = b_1, \ldots, b_n$, su distancia euclidiana se define como $D(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$.

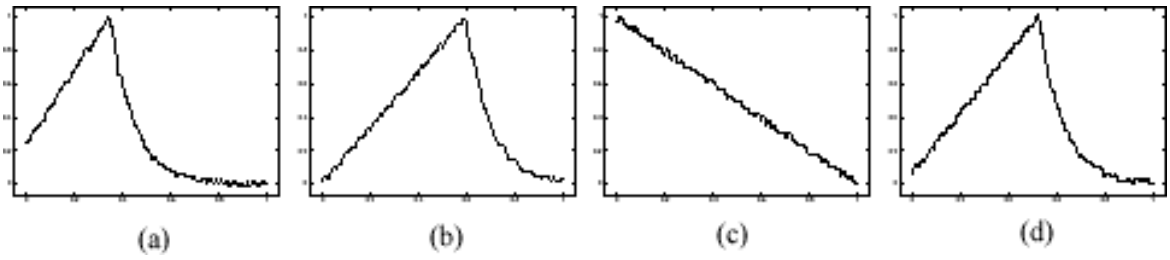

Figura 1.2.1: Noción de similitud de series temporales: bajo distancia euclidiana la curva (a) se encuentra más cercana a la curva (c) y en la interpretación humana la curva (a) se encuentra más cercana a (b) y (d) (TASDA, 2002).

Dado que la interpretación de los resultados es un asunto importante en el descubrimiento del conocimiento, la escogencia de parámetros de similitud es muy importante y no se escoge apropiadamente por los enfoques estáticos (TASDA, 2002).

1.2.3. Ambigüedad de características de baja granularidad

Es deseable la introducción de características de baja granularidad desde el punto de vista de la interpretación de los datos. Sin embargo, aunque los altos niveles de abstracción son útiles para la reducción de los datos y la interpretación, la información reducida es a menudo ambigua y depende del contexto; un usuario puede interesarse en diferentes aspectos según sea este. Pocos algoritmos de descubrimiento del conocimiento y minería de datos toman en cuenta esta ambigüedad explícitamente en el proceso. Además es necesario notar que las abstracciones son muchas veces heurísticas y casi nunca son cuestionadas luego (TASDA, 2002). Si se toma en cuenta, por ejemplo, el horario de trabajo de una persona, es más simple para nosotros como humanos saber que esa persona llegó tal día a trabajar o que esa persona no faltó a trabajar en todo el mes. Sin embargo, aunque esta reducción

permite manejar con más facilidad la información o interpretarla más fácilmente, puede mermar la inferencia que se podría hacer del comportamiento del trabajador. Ya que podría ser muy útil, saber con exactitud al segundo, si esa persona hace demasiadas pausas como tomar café, entre otras cosas.

Lo anteriormente expuesto se puede ver ilustrado en la serie temporal de la figura 1.2.2. Se pueden notar dos picos mayores, los cuales justo antes de decrecer, tienen un pequeño pico (señalado en la figura). Es justo entonces preguntarse si esto es importante o es una coincidencia. Además, también es válido preguntar si se va a permitir que un procedimiento heurístico de conversión de la serie temporal decida sobre esto. La respuesta sería que mejor no, pues si los picos pequeños son eliminados equivocadamente ningún algoritmo de detección de patrones será capaz de recuperar su importancia (TASDA, 2002).

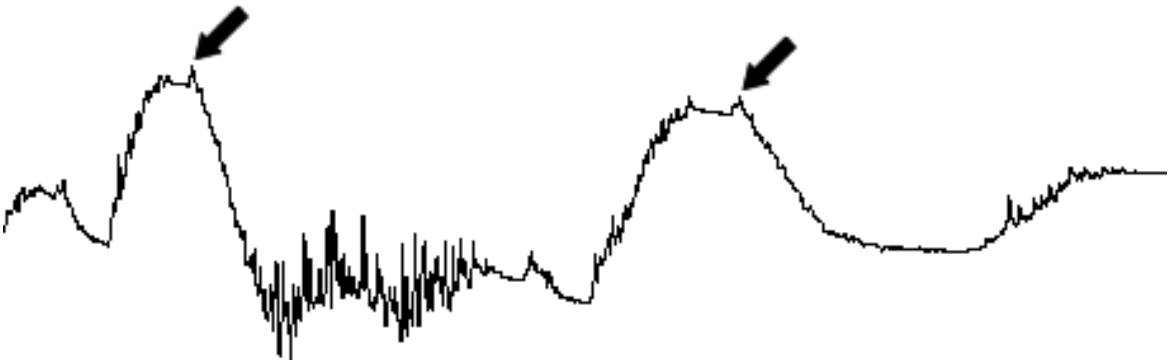

Figura 1.2.2: Serie temporal numérica (TASDA, 2002).

Aún así, no se querrá considerar cada dato puntual por separado, dado que esto elevaría significativamente el costo computacional del proceso de descubrimiento de patrones.

1.2.4. Medida de interés

En el paso de descubrimiento de conocimiento del proceso de minería de datos se buscan, en el espacio de patrones, hipótesis con el fin de encontrar relaciones de interés. A pesar de que en tal momento sería deseable inspeccionar la completitud de los patrones, muy posiblemente el número de patrones será demasiado grande para su inspección. Sin embargo, el problema de encontrar una medida de interés no es específico a la minería de datos temporales. El hecho de lidiar con datos temporales permite nuevas maneras de atacar el problema (TASDA, 2002).

Generalmente se buscan propiedades de subsecuencias significativas y se caracterizan eventos reduciendo el contenido de información mientras se desarrolle el evento, ya que al inicio de un evento no se está seguro de su continuación, más cuanto más tiempo se observa, se obtiene mayor seguridad acerca del estado actual del sistema y se vuelve más fácil predecir la siguiente observación. Este concepto únicamente es útil para datos de tipo secuencial/temporal pero no para datos que no son temporales (TASDA, 2002). Esta característica justifica la necesidad de un enfoque más sistémico y especializado en el descubrimiento de patrones en los datos temporales.

1.3. Importancia social

El efecto de los patrones secuenciales, previamente extraídos vía minería de datos, en la toma de decisiones es de singular importancia. Esto se debe a que muchas transacciones de negocios, registros de telecomunicaciones, datos climáticos, y procesos de producción son datos de secuencias temporales. Además, la minería con estos datos es útil para mercadeo, retención de clientes, predicción de clima y otros (Han & Kamber, 2001). Incluso aprender a predecir subsecuencias de even-

tos poco frecuentes pero altamente relacionadas puede ser útil en ataques a redes de computadoras o transacciones fraudulentas en instituciones financieras (Vilalta & Ma, 2002). Otro campo de aplicación es la medicina, donde es posible encontrar relaciones entre síntomas y diagnósticos en el tiempo, con lo cual se podrían crear planes de medicina preventiva específicos y locales. Son muchos los campos de aplicación tal como se verá en los antecedentes y trabajo relacionado (sección 2.1).

Así, tomar decisiones racionales requiere de conocimiento de los fenómenos en el dominio de aplicación y los patrones secuenciales juegan un rol importante. La minería de patrones secuenciales en grandes bases de datos de transacciones de clientes se enfocó primero dirigida a proveer información de valor a los negocios, tal como patrones de compra o tendencias de inventario. La minería de patrones secuenciales generalizados ha sido una solución más eficiente. Las características temporales en los modelos o procesos de minería pueden proveer información precisa acerca de dominios de negocios en evolución continua, pero pueden también beneficiar otras áreas de aplicación (Letia, Craciun, Köpe & Lelutiu, 2000) como las ya mencionadas. Es por esto que el modelo generado debe ser eficaz en encontrar los patrones ocultos en los datos.

1.4. Objetivos y alcance

Manteniendo la importancia de que el modelo de minería de patrones secuenciales debe ser aplicable o eficaz con la utilización de lógica temporal, se plantearon los siguientes objetivos y alcance.

Objetivo general

Aplicar la lógica modal temporal en el descubrimiento de patrones secuenciales en bases de datos y crear un nuevo modelo de descubrimiento de patrones secuenciales basado en la lógica temporal.

Objetivos específicos

- Escoger entre los fundamentos teóricos de la lógica temporal aquellos que sean aplicables al descubrimiento de patrones secuenciales.
- Diseñar e implementar un modelo para el descubrimiento de patrones secuenciales en bases de datos utilizando lógica temporal.
- Ejecutar el modelo de descubrimiento de patrones secuenciales sobre un conjunto seleccionado de casos de prueba en bases de datos relacionales.
- Analizar y documentar los resultados de la ejecución del modelo de descubrimiento de patrones secuenciales sobre los casos de prueba.

Alcance y limitaciones

Los datos temporales pueden estar representados de diferentes maneras: series de tiempo, fragmentos de series de tiempo, secuencias de eventos y secuencias de estado o de intervalo (TASDA, 2002). Este trabajo se limitó al descubrimiento de patrones secuenciales, que es la minería de datos que trabaja sobre los datos temporales representados como secuencias.

Existen también diferentes modelos para las bases de datos. Entre ellos podemos encontrar las bases de datos jerárquicas, las bases de datos relacionales, las bases de

datos orientadas a objetos u otras. En esta investigación el enfoque fue únicamente sobre las bases de datos relacionales.

Además, como técnica se utilizó el razonamiento lógico-temporal, sin embargo, la lógica utilizada fue del tipo monotónica. La propiedad de monotonicidad es incompatible con algunas de las maneras naturales de pensamiento. Por ejemplo, si se dice que todos los pájaros vuelan, se concluirá que algunos pájaros en particular vuelan, pero luego nos damos cuenta que los avestruces no vuelan. Agregar este nuevo hecho puede bloquear la conclusión anteriormente hecha. Lidiar con este tipo de problemas requiere de lo que llamamos lógicas no monotónicas. No se discutió este tipo de lógica en esta tesis pues están rodeadas de especulación y turbulencia (Winston, 1984), con lo cual se pudo entorpecer los objetivos básicos de esta investigación.

Este trabajo tampoco se enfocó en producir un algoritmo escalable. Dicho de otra manera, no se enfatizó en producir un modelo donde el tiempo de corrida creciera en forma lineal con respecto a la cantidad de datos, sino únicamente en resolver el problema de encontrar patrones secuenciales utilizando lógica temporal.

Además, no se investigó para desarrollar una herramienta que produzca datos artificiales de prueba para alimentar la ejecución del programa realizado. Tal como lo muestran diferentes autores (Argüello Venegas, 1997; Cooper & Zito, 2007; Omari, 2008; Omari, Langer & Conrad, 2008), producir datos artificiales de prueba, sobre todo cercanos a la realidad, no es trivial y es un tópico de investigación en sí mismo.

1.5. Metodología

Para cumplir con los objetivos general y específicos, se llevaron a cabo las siguientes etapas:

- Estudio teórico
- Modelamiento e implementación del descubridor de patrones
- Ejecución del descubridor de patrones sobre casos de prueba
- Análisis y documentación de los casos de prueba

Cada una de estas etapas se discute en las siguientes sub-secciones.

1.5.1. Estudio teórico

El estudio teórico del descubrimiento de patrones secuenciales se hizo exponiendo el problema en el contexto de la lógica temporal. Fue necesario, entonces, encontrar un medio de transformar las bases de datos relacionales a una representación lógica sobre la cuál fuera posible ejecutar razonamiento lógico temporal.

En la etapa de estudio teórico se realizaron las siguientes actividades:

- Se analizó la materia sobre lógica modal y lógica temporal buscando aplicabilidad en patrones secuenciales
- Se hizo un estudio sobre datos secuenciales y bases de datos temporales.
- Se estudió en general la minería de bases de datos aplicable: minería de series temporales, minería de datos secuenciales y minería de patrones secuenciales.

- Se realizó también un estudio general de las áreas de bases de datos deductivas, programación lógica y programación lógica temporal.

Estos estudios se reflejan en el marco teórico.

1.5.2. Modelamiento e implementación del descubridor de patrones

El modelamiento del descubridor de patrones se hizo formalmente gracias al estudio teórico. Este modelo formal condujo a un diseño de programa para descubrir patrones. Un programa de computador se implementó en base al diseño realizado. Esta etapa exigió además plantear qué tipo de transformaciones debían aplicarse a los datos para poder ser utilizados por la herramienta generada.

Esta etapa se compuso de las siguientes actividades:

- Se formalizó el concepto de patrón secuencial, tal como se verá en la sección 2.6 del marco teórico, la formalización provista por De Amo, Junior, Giacometti y Clemente (2007) fue suficiente.
- Se analizaron mecanismos adecuados de transformación de bases de datos relacionales a bases de datos lógicas, tal como se menciona en la sección 3.1.2.
- Se analizó un único mecanismo de tratamiento del tiempo, la lógica temporal, la cual se describe en la sección 2.5 del marco teórico.
- Se diseñó un modelo de descubrimiento de patrones secuenciales por medio de lógica temporal, descrito en el capítulo 3.
- Se programó el modelo de descubrimiento de patrones secuenciales, el cual está disponible como *software* libre en Internet.

1.5.3. Ejecución del descubridor de patrones

Una vez implementado el descubridor de patrones, se ejecutó sobre una base de datos de prueba de una aplicación de la vida real. Esta ejecución fungió como etapa de realización de pruebas de la herramienta realizada.

Las siguientes actividades se realizaron en esta etapa:

- Se seleccionaron dos posibles conjuntos de datos para su estudio por medio del programa desarrollado y se escogieron ambos para las pruebas al ser de interés para la investigación. Estos conjuntos de datos están descritos en el capítulo 4.
- Se realizó la ejecución del programa. Dicha ejecución se discute en el capítulo 4.

1.5.4. Análisis y documentación de los casos de prueba

Para poder evaluar los patrones obtenidos por el descubridor de patrones fue necesario evaluar la eficacia del modelo diseñado.

El concepto de patrón secuencial es introducido para capturar comportamientos típicos en el tiempo, es decir, comportamientos suficientemente repetidos para ser relevantes para quién toma decisiones (Masseglia, Cathala & Poncelet, 1998). Por ello es necesario evaluar la eficacia.

Así, esta etapa se incluyeron las siguientes actividades:

- Se evaluaron las pruebas realizadas tal como se muestra en el capítulo 4.
- Se concluyeron resultados en la sección 5.1.

1.6. Contenido y contribuciones

Esta tesis se enfoca en la minería de datos temporales y presenta un modelo de minería de patrones secuenciales que incorpora la lógica temporal en la etapa de búsqueda patrones. Este resultado basa la minería de patrones secuenciales en un marco teórico sólido en lugar de fundamentarla únicamente en la utilización de estructuras de datos eficientes.

Se comienza la presentación de esta investigación con la introducción de los antecedentes y trabajo relacionado en el capítulo 2. Este capítulo también presenta la teoría sobre la cual se fundamenta esta investigación. Se introducen entonces la minería de series temporales, datos secuenciales y patrones secuenciales, la lógica modal, la lógica temporal, los patrones secuenciales en lógica temporal y la programación lógica temporal.

El capítulo 3 presenta un modelo para la minería de patrones secuenciales utilizando lógica temporal. Primero se introduce el modelo describiéndolo. Finalmente se detalla una implementación realizada del modelo describiendo las tecnologías utilizadas.

Se muestran dos casos de estudio en el capítulo 4. El primer caso de estudio es un caso ilustrativo cuya importancia es la de explicar el algoritmo diseñado. Le sigue luego un caso de estudio utilizando la implementación realizada en datos artificiales. En este capítulo también se presentan resultados.

Finalmente, en el capítulo 5, se dan conclusiones y se discute sobre el posible trabajo futuro.

En resumen, este trabajo de investigación generó las siguientes contribuciones a la comunidad académica costarricense:

- Una apropiación del conocimiento sobre lógica modal y lógica temporal.
- Un modelo nuevo de minería de patrones secuenciales con lógica temporal que incluye un algoritmo nuevo de minería de patrones secuenciales que modifica el paso de búsqueda de *Patrones Secuenciales Generalizados* (Agrawal & Srikant, 1994b, 1995b) sustituyéndolo por un conteo de fórmulas de lógica temporal.
- La realización de una aplicación de Software Libre/Código Abierto que es una prueba de concepto del nuevo modelo.
- La reutilización y apropiación tecnológica de diferentes componentes de Software Libre para la implementación de la aplicación de prueba de concepto.

Capítulo 2

Marco teórico

Este capítulo primero presenta los antecedentes y trabajo relacionado. Luego se introduce de manera breve a la minería en bases de datos con un enfoque temporal en la sección 2.2. En la sección 2.3 se definen los datos secuenciales. En las secciones 2.4 y 2.5 se presentan fundamentos teóricos de la lógica modal y de la lógica temporal. Por último, se presentan en las secciones 2.6 y 2.7 los patrones secuenciales en lógica temporal y la programación lógica temporal, respectivamente.

2.1. Antecedentes y trabajo relacionado

Los primeros reportes de investigación en minería de datos buscando patrones secuenciales, que dieron inicio a esta línea de investigación, comienzan en el seno de IBM en 1994 (Agrawal & Srikant, 1994b), cuando Agrawal y Srikant presentan como resolvieron el problema en grandes bases de datos de transacciones de clientes utilizando variantes de APRIORI (APRIORIALL y APRIORISOME). Cada transacción de cliente consistía a su vez de un identificador del cliente, el momento en que se efectuó la transacción y los elementos comprados en la transacción.

Efectivamente, una gran mayoría de autores coinciden en utilizar algoritmos con el principio APRIORI para el descubrimiento de patrones secuenciales. Este principio mantiene que si un patrón secuencial de tamaño k es infrecuente, su superconjunto (de tamaño $k+1$) no puede ser frecuente, lo cual permite recortar de manera eficiente el espacio de búsqueda[1]. Por lo tanto, la mayoría de los métodos para minería de patrones secuenciales utilizan variaciones de APRIORI, aunque pueden considerar diferentes parámetros y restricciones (Han & Kamber, 2001).

El trabajo de Agrawal y Srikant permitió encontrar patrones tales como el cliente renta "La guerra de las galaxias", luego "El imperio contraataca" y luego "El regreso del Jedi" en ese orden. La base de datos se transforma de transacciones a secuencias de conjuntos de elementos por cliente, y se minan los patrones máximos utilizando uno de dos algoritmos de tipo APRIORI. Su investigación es presentada al público en 1995 (Agrawal & Srikant, 1995a), y en el mismo año en IBM se presenta un reporte de investigación con generalizaciones y mejoras de rendimiento (Agrawal & Srikant, 1995b), el cual es presentado al público en 1996 (Srikant & Agrawal, 1996). Entre las mejoras presentadas (Srikant & Agrawal, 1996) se encuentra el algoritmo GSP para encontrar patrones secuenciales generalizados. La experimentación mostró que GSP es más rápido que el algoritmo APRIORIALL presentado anteriormente (Agrawal & Srikant, 1995a). GSP escala linealmente con el número de secuencias y tiene una muy buena propiedad de escalado con respecto del tamaño promedio de secuencia.

En el trabajo presentado por Agrawal y Srikant (1995a) y Srikant y Agrawal (1996), se utilizó para la prueba de los algoritmos un generador de datos sintéticos de IBM llamado Quest (Srikant, 1995). En un análisis realizado por el autor de esta tesis, se encontró que el código tiene errores importantes como referencias a

[1]Nótese además que esto es una aplicación concreta de una medida de interés tal como se menciona en la sección 1.2.4.

direcciones nulas. Además, la herramienta está escrita en C++ y muestra errores de sintaxis en los compiladores actuales de C++, por lo cual es necesario hacer cambios al código para poder correr la herramienta[2]. Por último, Cooper y Zito (2007), hacen una crítica importante a la generación de datos sintéticos por IBM Quest. Motivados por la afirmación (apoyada por evidencia empírica) de que la ocurrencia de elementos en una canasta de mercado sigue un patrón diferente, propusieron un modelo alternativo para generar datos artificiales.

Mannila, Toivonen y Verkamo (1995), presentan también en el año 1995 una investigación sobre el descubrimiento de episodios frecuentes en secuencias. Ellos consideran el problema de reconocer episodios frecuentes en secuencias de eventos. Un episodio se define como una colección de eventos que ocurren dentro de intervalos de tiempo de un tamaño determinado en un orden parcial determinado. Una vez que se conocen los episodios, se pueden producir reglas para describir o predecir el comportamiento de la secuencia. Se describe un algoritmo eficiente para el descubrimiento de todos los episodios frecuentes de una clase dada de episodios.

Padmanabhan y Tuzhilin, publican en 1996 sobre el descubrimiento de patrones con un enfoque de lógica temporal que se considera como una continuación al trabajo de Mannila et al. (1995). Inclusive, la investigación de Padmanabhan y Tuzhilin está mencionada en el trabajo de Mannila, Toivonen y Verkamo (1997). Mannila, Toivonen y Verkamo sostienen que el formalismo de la propuesta de Padmanabhan y Tuzhilin es fuerte y permite expresar patrones más complejos que los episodios, pero no es clara cuál es la complejidad de diferentes tareas de descubrimiento. Efectivamente, la conclusión de Padmanabhan y Tuzhilin es que los resultados muestran que el rendimiento de su implementación es muy lento (esto se debe en parte a

[2]Actualmente con los cambios realizados es posible correr el programa en los siguientes compiladores: Visual C++ Express 2007, GNU C++, OpenWatcom y Borland C++ 5.5. Estos cambios se encuentran documentados en el apéndice F.

que simularon algunas expresiones haciendo el programa más complejo). Inclusive ellos proponen la necesidad de desarrollar intérpretes de programación lógica temporal que apoyen características importantes como la ejecución pseudo-paralela de reglas temporales y que provean un apoyo eficiente para la dimensión temporal. Así, la línea de investigación de Padmanabhan y Tuzhilin no tiene continuación directa (Fürnkranz, 1997; Rainsford & Roddick, 1999; Kam, 2000; Ramirez, Cook, Peterson & Peterson, 2000b, 2000a; Wang, Yang & Yu, 2001; Roddick, Spiliopoulou & Society, 2002; Yang, Wang & Yu, 2003; Mooney & Roddick, 2004; Liu, Xiong, Das-Gupta & Zhang, 2006; Tseng & Lin, 2007; Gregory & Shneiderman, 2010), lo cual presenta una oportunidad de investigación que se aprovecha en este trabajo.

Dehaspe y Toivonen, proponen en 1999 el descubrimiento de patrones Datalog frecuentes. Aunque no utilizan lógica temporal y no está en el artículo indicado que es posible utilizarla, los patrones o las consultas Datalog podrían estar expresados en lógica temporal.

Sandra de Amo et al. muestran en el año 2004 el paralelismo entre el algoritmo APRIORI y los patrones secuenciales expresados en lógica temporal. Es decir, muestran como las secuencias de elementos frecuentes (obtenidas con APRIORI) son reglas de lógica proposicional temporal y hacen la generalización a primer orden. Sin embargo, sus algoritmos no incluyen este tipo de lógica. Esta lógica únicamente se incorpora en la expresión del resultado.

2.2. Minería en bases de datos

Ramakrishnan y Grama (1999) nos dicen que los objetivos comunes de todas las aplicaciones de minería de datos son la detección, la interpretación y la predicción de patrones cuantitativos y cualitativos en los datos.

Otros como Zhao y Bhowmick (2003) consideran a la minería de datos como el proceso de extraer información o patrones interesantes (no triviales, implícitos, previamente desconocidos y potencialmente útiles) desde grandes repositorios tales como: bases de datos relacionales, bodegas de datos, repositorios XML y otros.

Algunos consideran a la minería de datos como la etapa principal y una etapa más de un proceso global llamado descubrimiento de conocimiento en bases datos (Hätönen, Klemettinen, Mannila, Ronkainen & Toivonen, 1996). Luego de la minería de datos siguen dos etapas que son el post-procesamiento del conocimiento descubierto (selección de los patrones realmente interesantes, presentación de los patrones, ...) y la utilización el conocimiento descubierto (Hätönen et al. 1996).

2.2.1. Minería de series temporales y datos secuenciales

Los datos secuenciales y las series temporales son secuencias de valores o eventos ordenados. Usualmente los valores son tomados en intervalos iguales de tiempo. Este tipo de bases de datos son utilizadas en muchas aplicaciones, tales como el estudio de las fluctuaciones diarias de un mercado bursátil, rastros de un proceso de producción dinámica, experimentos científicos, tratamientos médicos y otros. Además, las series temporales también son secuencias. Sin embargo, las secuencias son cualquier base de datos que consiste en secuencias de eventos ordenados, quedando opcional la noción concreta de tiempo. Por ejemplo, las secuencias de paso por páginas Web son datos secuenciales, pero no necesariamente datos de series temporales (Han & Kamber, 2001). A estos tipos de base de datos podemos aplicar diferentes tipos de minería tales como análisis de tendencias, búsqueda de similitud, minería de patrones secuenciales y minería de patrones en datos relacionados con el tiempo.

Esta investigación únicamente se ocupó de la minería de patrones secuenciales cuya definición se da en la siguiente sección.

2.2.2. Minería de patrones secuenciales

La minería de patrones secuenciales es la búsqueda de patrones que ocurren frecuentemente relacionados con el tiempo o con otras secuencias. Un ejemplo de patrón secuencial, dado en la introducción, es "*Un cliente que compró una computadora personal Pentium nueve meses atrás probablemente va a ordenar un chip CPU dentro de un mes*" o "*El cliente renta* La guerra de las galaxias, *luego* El imperio contraataca *y luego* El regreso del Jedi". Inclusive, como se decía en la introducción, dado que muchas transacciones de negocios, registros de telecomunicaciones, datos climáticos y procesos de producción son datos de secuencias temporales, la minería de patrones secuenciales es útil en el análisis de tales datos para objetivos de mercadeo, retención de clientes, predicción climática y otros. Es necesaria, entonces, una definición de patrón secuencial más formal.

2.3. Datos secuenciales

De manera general una secuencia es una lista ordenada, la cuál puede ser de artículos, eventos, valores u otros. De manera informal las secuencias se diferencian de las series en que las secuencias pueden tener asociados más que solamente valores numéricos y que además los elementos de la secuencia no están separados necesariamente por intervalos iguales en el tiempo, el orden es parcial. Incluso puede ser que no se dé un valor a la variable tiempo sino que solamente un elemento ocurre antes de otro, sin importar cuando. Como ejemplo ilustrativo, Dunham (2002) da la siguiente definición: sea $I = \{I_1, I_2, \ldots, I_m\}$ un conjunto de artículos. Una **secuencia**, S, es: $S = \langle s_1, s_2, \ldots, s_n \rangle$, donde $s_i \subseteq I$.

Además, se puede notar que las definiciones de secuencias dependen mucho del objeto de estudio, por ejemplo, donde el objeto de estudio son eventos Vilalta y Ma (2002) define las secuencias así:

> Una secuencia de eventos es una colección ordenada de eventos $D = \langle d_1, d_2, \ldots, d_n \rangle$, donde cada evento d_i es un par $d_i = (e_i, t_i)$. El primer elemento del par, e_i, indica el tipo de evento. El segundo elemento del par, t_i, indica su ocurrencia en el tiempo.

Estas dos ilustraciones anteriores llevan a preguntar cuál es la definición de dato secuencial que se utilizó en este trabajo de investigación. Las secuencias se expresaron en lenguaje lógico temporal. Es decir, dado que la lógica temporal permite expresar orden o secuencias, la definición de dato secuencial es la clase de todas las secuencias expresables por medio de fórmulas de lógica del tiempo. Sin embargo, en esta investigación se estudió una sola clase de fórmula del tiempo (o patrón). La lógica temporal se explica a continuación en este marco teórico en el apartado 2.5. La clase de fórmula o secuencia temporal que se estudió y su definición se da más adelante en la sección 2.6.

2.4. Lógica modal

Los seres humanos efectuamos diferentes tipos de razonamiento, por ello es necesario tener diferentes lógicas. Actualmente, entre los diferentes tipos de lógica que existen se pueden mencionar: el razonamiento matemático, el razonamiento con sentido común, el razonamiento abductivo y de diagnóstico, el razonamiento inductivo, la lógica difusa, las lógicas multivaluadas, la lógica probabilística, las lógicas intuicionista y constructivista, las lógicas modales, la lógica paraconsistente y la lógica deóntica (Bertossi, 1996), entre otras. Dado que la lista anterior no es una

lista exhaustiva se puede concluir que "hay diversas formas de razonamiento y podemos esperar que surjan lógicas *ad hoc* que las modelen" (Bertossi, 1996). En esta sección, se introduce la lógica modal que es el tipo general de lógica que se utilizó.

Las lógicas modales modelan el razonamiento a través de modalidades debido a ciertas limitaciones de las lógicas de predicados y la lógica clásica. Típicamente se utilizan las modalidades "necesario" y "posible". Si se denotan estas dos modalidades con los operadores L y M, respectivamente, y p es una proposición, entonces, podemos hacer aseveraciones de la forma Lp y Mp que se leen como "necesariamente p" y "posiblemente p". Tal lógica tiene entre sus axiomas lógicos, o sea, entre lo que se acepta como verdadero por simple lógica, proposiciones de la forma $Lp \rightarrow p$, $L(p \rightarrow q) \rightarrow (Lp \rightarrow Lq)$, $Mp \longleftrightarrow \neg L \neg p$. Por ejemplo, el último axioma muestra la relación que existe entre ambos operadores y significa que es lo mismo plantear que "posiblemente p" que decir que "no necesariamente no p" (Bertossi, 1996).

En ciencias de la computación han surgido diversas lógicas modales, por ejemplo la lógica dinámica y la lógica temporal. Esta última es la de interés en este trabajo y se ve más adelante en este capítulo. Además, en el área de representación de conocimiento en inteligencia se han introducido lógicas con las modalidades de creencia B (por "belief[3]") y conocimiento K (por "knowledge[4]") (Bertossi, 1996).

Existe una notación simbólica ($\Box$ y $\Diamond$) que se utiliza típicamente en lugar de los operadores textuales L y M, y que es común a otros tipos de lógica modal como la lógica temporal. En la sección siguiente se describe esta notación y su semántica. Más adelante se describe la lógica modal llamada temporal.

[3]Inglés para creencia.
[4]Inglés para conocimiento.

2.4.1. Lógica monomodal

La semántica habitual de los sistemas de lógica modal requiere una estructura relacional: un universo y relaciones sobre el mismo. En el caso más simple, la lógica modal básica, es suficiente un universo y una relación binaria. Es decir, intuitivamente, para evaluar la satisfacción de una fórmula se requiere interpretarla sobre un grafo dirigido (Fernández, Manjarrés Riesco & Diéz Vegas, 2007). Más precisamente, cada fórmula se puede evaluar sobre cada nodo del grafo, satisfaciéndose quizá en algunos nodos y en otros no. Esta evaluación local es lo que caracteriza a este tipo de lógicas.

Las lógicas modales son una herramienta para analizar estructuras relacionales. En ciencias de la computación éstas son las estructuras sobre las que se modela prácticamente toda la actividad del campo (sistemas de transiciones etiquetadas, autómatas, redes, agentes y otros). La elección de los operadores modales adecuados, siempre con un enfoque semántico común, permite proponer sistemas lógicos específicos para ciertas actividades (Fernández et al. 2007).

Desde el punto de vista formal, es interesante observar cómo las fórmulas modales pueden traducirse sistemáticamente a fórmulas de Primer Orden. También cómo definen, con una sintaxis sencilla y una evaluación local, propiedades globales de la estructura.

Se presenta, entonces, la lógica modal como un lenguaje alternativo mediante el cual se describen y analizan estructuras relacionales con base en el trabajo de Fernández et al. (2007).

2.4.1.1. Lógica modal básica

Se presenta el lenguaje modal más simple, el cual se construye sobre el lenguaje de la lógica de proposiciones, con la incorporación de dos nuevos símbolos. Como es usual la sintaxis del lenguaje facilita el conjunto de fórmulas y de expresiones correctamente formadas.

Dada una fórmula de la lógica modal básica, mediante su semántica se comprueba si es verdadera o no en una determinada estructura relacional. De manera específica, las estructuras sobre las que se evalúa una fórmula de este lenguaje constan de una única relación binaria.

Alfabeto y lenguaje

Definición 2.4.1. (Alfabeto) El alfabeto de la lógica modal básica consta de los siguientes elementos (Fernández et al. 2007):

1. letras proposicionales: $p_0,\ p_1,\ p_2,\ \ldots$
2. símbolos lógicos[5]:
   - 2.1. constantes proposicionales: $\bot, \top$
   - 2.2. conectivas monarias ($\neg$) y binarias: $\wedge, \vee, \rightarrow$
   - 2.3. operadores modales: $\Box, \Diamond$
3. símbolos auxiliares: '(' y ')'

Definición 2.4.2. (Fórmulas del lenguaje modal básico) Las fórmulas del lenguaje se definen por la siguiente expresión BNF (Fernández et al. 2007):

[5]Se utiliza el símbolo $\leftrightarrow$ como abreviatura. De esta manera, $(\phi \leftrightarrow \psi)$ abrevia la fórmula $((\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi))$.

$$\phi ::= p \mid \bot \mid \top \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid \Box\phi \mid \Diamond\phi$$

Es decir, una fórmula es, cualquier cadena que se genere por aplicación finita de las siguientes reglas[6]:

1. Cada letra proposicional es una fórmula
2. Las constantes proposicionales $\bot$ y $\top$ son fórmulas
3. Si ψ es una fórmula, entonces $\neg\psi$ es una fórmula
4. Si ψ y χ son fórmulas, entonces son fórmulas $(\psi \wedge \chi)$, $(\psi \vee \chi)$ y $(\psi \rightarrow \chi)$
5. Si ψ es una fórmula, entonces son fórmulas $\Box\psi$ y $\Diamond\psi$

Subfórmulas Al igual que en la lógica de proposiciones o de predicados, las demostraciones sobre el lenguaje utilizan el principio de inducción estructural. Las definiciones se producen recursivamente. La siguiente definición es un ejemplo de ello.

Definición 2.4.3. (Conjunto de subfórmulas) A cada fórmula ϕ le corresponde un único conjunto de subfórmulas $Subf(\phi)$. Este conjunto de subfórmulas se define como (Fernández et al. 2007):

- $Subf(\Box\phi) = \{\Box\phi\} \cup Subf(\phi)$,
  $Subf(\Diamond\phi) = \{\Diamond\phi\} \cup Subf(\phi)$
- $Subf((\phi * \psi)) = \{\phi * \psi\} \cup Sub(\phi) \cup Sub(\psi)$
  para toda conectiva binaria *

[6] **Notación:** Se denota *Var* al conjunto de las letras proposicionales y *Form* al conjunto de las fórmulas.

- $Subf(\neg\phi) = \{\neg\phi\} \cup Subf(\phi)$
- $Subf(\bot) = \{\bot\}$,
  $Subf(\top) = \{\top\}$
- $Subf(p) = \{p\}$
  para cada letra proposicional

Sustituciones La sustitución uniforme permite escribir una fórmula a partir de otra. Formalmente, una sustitución es una función del conjunto de letras proposicionales en el conjunto de fórmulas: $\sigma : Var \mapsto Form$. Por medio de esta función se puede definir otra que asigna a cada fórmula su transformada (por una cierta sustitución): $()^\sigma : Form \times Sust \mapsto Form$.

Definición 2.4.4. (Instancia, por sustitución, de una fórmula) Dada una sustitución $\sigma : Var \mapsto Form$, la transformada ϕ^σ de una fórmula ϕ se define como (Fernández et al. 2007):

1. $(\Box\phi)^\sigma = \Box\phi^\sigma$,
   $(\Diamond\phi)^\sigma = \Diamond\phi^\sigma$
2. $(\phi * \psi)^\sigma = (\phi^\sigma * \psi^\sigma)$
   para toda conectiva binaria *
3. $(\neg\phi)^\sigma = \neg\phi^\sigma$
4. $\bot^\sigma = \bot$,
   $\top^\sigma = \top$
5. $p^\sigma = \sigma(p)$,
   para cada letra proposicional

La fórmula ϕ^σ es la instancia de ϕ por la sustitución σ.

Instancias, por sustitución, de tautologías De especial interés son las sustituciones sobre tautologías clásicas proposicionales (o sea, no modales). Por ejemplo, la transformación de $(p \vee \neg p)$ en:

$$(\Box(q \rightarrow \Diamond r) \vee \neg\Box(q \rightarrow \Diamond r))$$

Semántica relacional Las fórmulas de la lógica modal se interpretan sobre objetos matemáticos denominados modelos. Un modelo (definición 2.4.7) se define a partir de un marco (definición 2.4.5) y de una asignación (definición 2.4.6), que precisa qué letras proposicionales son ciertas en cada nodo.

Dada una fórmula ϕ y un modelo, se define (en 2.4.9) cuándo 'ϕ se satisface (es verdadera) en un nodo determinado del modelo'. Sobre esta semántica se consideran los conceptos usuales de validez, equivalencia y consecuencia lógica.

Marcos

Definición 2.4.5. (Marco) Un marco F (para la lógica modal básica) es un par $\langle W, R\rangle$, donde W es un conjunto no vacío y R es una relación binaria sobre W (Fernández et al. 2007).[7]

Conjuntos o familias de marcos Algunos resultados teóricos serán válidos no sólo para un cierto marco, sino para todos los de un determinado conjunto. Por ejem-

[7] **Notación:** Dependiendo del contexto o de la aplicación, los elementos de W se denominan mundos, mundos posibles, estados, instantes, ... Se utiliza indistintamente 'mundos', 'estados' y 'nodos'.

Cuando el mundo w_1 esté relacionado con w_2 diremos que 'desde w_1 se accede a w_2' o que 'w_2 es accesible desde w_1' y se denota como Rw_1w_2. A la relación R se le denomina *relación de accesibilidad*.

En la notación se han mantenido las iniciales en inglés de los conceptos: $F = \langle W, R\rangle$ corresponde a $Frame = \langle Worlds, Relation\rangle$.

Un marco es, de manera general, *una estructura relacional*: un conjunto y relaciones diversas n-arias sobre el mismo conjunto. En este caso, para interpretar la lógica modal básica sólo se necesita una relación binaria.

plo, 'todos los marcos con relación reflexiva' o 'los que sean reflexivos y transitivos'. Se utiliza la notación $\mathcal{F}$ para referirse a un conjunto (no vacío) de marcos.

Modelos

Definición 2.4.6. (Asignación) Una asignación v en un marco $\langle W, R\rangle$ es una función $v : Var \mapsto \mathcal{P}(W)$ que asocia a cada letra proposicional el subconjunto de mundos donde es verdadera (Fernández et al. 2007).

Definición 2.4.7. (Modelo) Un modelo M (para la lógica modal básica) es una terna $\langle W, R, v\rangle$, donde $\langle W, R\rangle$ es un marco y v es una asignación sobre el mismo marco. Se denota también como $\langle F, v\rangle$ (Fernández et al. 2007).

El término modelo, en lógica de primer orden, se aplica a toda estructura sobre la que se satisface una fórmula. En lógica modal, se denomina modelo a la estructura matemática sobre la que se interpreta la fórmula, con independencia de su satisfacción.

Conjuntos o familias de modelos Algunos resultados teóricos son aplicables a todos los modelos de un determinado conjunto. Se utiliza la notación $\mathcal{M}$ para designar a un conjunto de modelos.

Definición 2.4.8. (Conjunto de modelos sobre un marco) Dado un marco $F = \langle W, R\rangle$, cada asignación distinta sobre el mismo produce un modelo diferente. Si se consideran todas las asignaciones posibles, cada marco determina un conjunto de modelos (Fernández et al. 2007).

Satisfacción La definición de una semántica, en un sistema lógico, permite responder a la pregunta ¿se satisface (es verdadera) la fórmula ϕ, interpretada sobre el objeto matemático O? Para el sistema aquí discutido:

¿Se satisface la fórmula ϕ, interpretada en el mundo w del modelo M?

Para evaluar fórmulas complejas se necesita una definición precisa y formal del concepto de satisfacción.

Definición 2.4.9. (Satisfacción) La satisfacción de una fórmula ϕ en un mundo w del modelo M, denotada como $M, w \Vdash \phi$, se define recursivamente tal como en el siguiente cuadro (Fernández et al. 2007):

1a	$M, w \Vdash \bot$		en ningún caso
1b	$M, w \Vdash \top$		en todo caso
1c	$M, w \Vdash p$	si y sólo si	$w \in v(p)$
2a	$M, w \Vdash \neg\phi$	si y sólo si	no $M, w \Vdash \phi$
2b	$M, w \Vdash (\phi \wedge \psi)$	si y sólo si	$M, w \Vdash \phi$ y $M, w \Vdash \psi$
2c	$M, w \Vdash (\phi \vee \psi)$	si y sólo si	$M, w \Vdash \phi$ ó $M, w \Vdash \psi$
2d	$M, w \Vdash (\phi \rightarrow \psi)$	si y sólo si	no $M, w \Vdash \phi$ ó $M, w \Vdash \psi$
3a	$M, w \Vdash \Box\phi$	si y sólo si	para todo $w' \in W$, si Rww' entonces $M, w' \Vdash \phi$
3b	$M, w \Vdash \Diamond\phi$	si y sólo si	existe un $w' \in W$ tal que Rww' y $M, w' \Vdash \phi$

Cuadro 2.4.1: Definición recursiva de satisfacción $M, w \Vdash \phi$

Observe cómo la satisfacción de una fórmula compleja depende recursivamente de la satisfacción de sus subfórmulas, hasta llegar al caso base: las fórmulas atómicas l[a-c].

- Las líneas l[a-c] y 2[a-d] son una copia de la semántica de la lógica proposicional. Definen cómo interpretar, en un mundo, una fórmula sin operadores modales: simplemente se procede como ya se hacía en lógica proposicional (considerando qué letras proposicionales son verdaderas en ese mundo).
- En la línea 2a. conviene fijarse en que, coloquialmente, el 'no' metalingüistico externo se convierte en un 'no' interno del lenguaje ($\neg$) y viceversa. Se abrevia 'no $M, w \Vdash \phi$' como $M, w \nVdash \phi$.
- Informalmente, 3a se puede ver desde una perspectiva procedimental: 'para verificar que $\Box\phi$ se satisface en un mundo, se inspeccionan todos sus mundos accesibles y se comprueba que en cada uno (sin excepción) se satisface la subfórmula ϕ'.
- La misma perspectiva se puede ofrecer de 3b: 'para verificar que $\Diamond\phi$ se satisface en un mundo, inspeccione sus mundos accesibles hasta encontrar al menos uno donde se satisface la subfórmula ϕ'.

La satisfacción es el concepto clave de la lógica modal.

Definición 2.4.10. (Operadores duales) Para todo operador modal $\triangle$ que se defina se considera el operador $\neg\triangle\neg$ al cual se denomina operador *dual* (Fernández et al. 2007).

Los operadores $\Diamond$ y $\Box$ son duales el uno del otro. Obsérvese que, con la semántica definida $M, w \Vdash \phi$ si y sólo si para todo w' accesible desde w se satisface ϕ, si y sólo si no existe un w' accesible desde w donde no se satisfaga ϕ, si y sólo si no $M, w \Vdash \Diamond\neg\phi$ y si y sólo si $M, w \Vdash \neg\Diamond\neg\phi$.

El operador $\Diamond$ tiene un sentido existencial (sobre el dominio de mundos localmente relacionados), mientras que el operador $\Box$ tiene un sentido universal. Se po-

dría haber definido el lenguaje con uno sólo de los dos operadores y considerar el otro como una abreviatura. Así, el lenguaje de la lógica modal básica es monomodal: sólo requiere definir un operador modal.

En general, los operadores modales que se podrán definir, ocurren por pares duales. A veces, ambos símbolos se definen como operadores primitivos del alfabeto, si no, se suele definir un símbolo adicional como abreviatura del dual del primitivo.

Conceptos semánticos básicos En esta sección se definió recursivamente la relación de satisfacción. En base a ella se definen otros conceptos semánticos fundamentales como *verdad*, *validez*, *consecuencia* o *equivalencia*. Los conceptos de validez, consecuencia y equivalencia se definen en el apéndice A que profundiza más en la lógica temporal.[8] El concepto de verdad se define en seguida.

Definición 2.4.11. (Fórmula verdadera) Si $M, w \Vdash \phi$, es decir, si una fórmula se satisface en un estado w de un modelo M diremos que es *localmente verdadera* o *verdadera* en ese estado (Fernández et al. 2007).

Lecturas del operador modal Se pretende formalizar el concepto de necesidad y es factible hacerlo sobre la lógica modal presentada si se considera que 'algo es necesario en un mundo (una situación) si se satisface en todo mundo accesible desde él'. Informalmente, allá hasta donde se puede mirar se produce p, luego desde esta situación p es necesario. En este contexto, una fórmula como $\Box p$ se puede leer como 'es necesario p'.

Como $\Diamond p$ equivale a $\neg\Box\neg p$, debiera leerse como 'no es necesario que no se satisfaga p'; es decir: 'es posible p'.

[8] En el apéndice A también se introducen los conceptos de teoría de la correspondencia y lógicas normales.

La semántica modal definida hasta el momento parece ajustarse a la modelización de este concepto. No es extraño, pues gran parte del trabajo en lógica monomodal se hizo con esta lectura en mente.

Cuando se desarrolla en detalle esta modelización se encuentra una objeción evidente: ¿Qué ocurre en un mundo no relacionado consigo mismo? Puede satisfacerse $\Box p$ sin que se satisfaga p en ese mundo. Con la lectura propuesta, se puede admitir que p es necesario en una situación sin que se verifique en la misma.

La adecuación del formalismo se produce, sin más sorpresas, si se requiere que la interpretación de este concepto se produzca sobre marcos reflexivos. Es decir, si se restringe a la lógica KT (definición A.3.1). Todas las fórmulas que se pueden demostrar partiendo de estos axiomas serán válidas en marcos formales, y todas ellas expresan relaciones aceptables entre los conceptos de necesidad ($\Box$), posibilidad ($\Diamond$) y expresiones lógicas proposicionales.

2.4.2. Lógicas polimodales

El lenguaje utilizado hasta este punto es monomodal: contiene un único operador modal $\Box$, junto a su operador dual $\Diamond$. Una gran parte de las aplicaciones de interés requieren lenguajes con más operadores modales.

Definición 2.4.12. (Fórmulas de un lenguaje polimodal) Se parte de un alfabeto proposicional que incluye el siguiente conjunto de operadores modales $\{[a], \ldots, [b], \langle a \rangle, \ldots, \langle b \rangle\}$. Las fórmulas del lenguaje son, exclusivamente, todas aquellas que se pueden generar por aplicación finita de las siguientes reglas (Fernández et al. 2007):

1. Cada letra proposicional es una fórmula

2. Las constantes proposicionales $\bot$ y $\top$ son fórmulas

3. Si ψ es una fórmula, entonces $\neg\psi$ es una fórmula
4. Si ψ y χ son fórmulas, entonces son fórmulas $(\psi \wedge \chi)$, $(\psi \vee \chi)$ y $(\psi \rightarrow \chi)$
5. Si ψ es una fórmula, entonces son fórmulas $[k]\psi$ y $\langle k \rangle\, \psi$, para cada uno de los operadores modales

Usualmente, no suelen incluirse los operadores duales como símbolos primitivos del alfabeto. Se les asigna un símbolo posterior, como abreviatura.

Semántica Como en la sección anterior, comiéncese considerando un lenguaje con dos únicos operadores $[a]$ y $[b]$ (junto a sus duales).

- Interpretar, en un mundo w, una fórmula como $[a]p$ o como $\langle a \rangle\, p$ requiere inspeccionar el valor de p en sus mundos a-relacionados
- Interpretar, en un mundo w, una fórmula como$[b]p$ o como $\langle b \rangle\, p$ requiere inspeccionar el valor de p en sus mundos b-relacionados
- Los mundos a-relacionados con w pueden ser distintos de sus mundos b-relacionados

Es decir, un lenguaje con dos operadores modales (junto a sus duales) requiere un marco $\langle W, R_a, R_b \rangle$ con dos relaciones binarias.

En el caso general, con n operadores modales, basta modificar la definición previa de satisfacción monomodal:

3'a $M, w \Vdash \langle k \rangle\, \phi$ si y sólo si existe un $w' \in W$ tal que $R_k w w'$ y $M, w' \Vdash \phi$

3'b $M, w \Vdash [k]\phi$ si y sólo si para todo $w' \in W$, si $R_k w w'$ entonces $M, w' \Vdash \phi$

Observe que esta definición exige que cada operador modal tenga asignada un relación binaria en el marco sobre el que se interpreta.

Interdependencia entre relaciones Existen otras muchas formas de definir una relación R_b a partir de una R_a. Se puede requerir, por ejemplo, que dos mundos estén relacionados en un sentido por R_a si y sólo si lo están en el otro por R_b. Es decir, que R_a y R_b, sean relaciones recíprocas o inversas. Más generalmente: ciertas aplicaciones pueden requerir *alguna dependencia entre las relaciones* de los marcos donde se interpretan.

La formulación de estas dependencias se puede expresar en lógica de predicados. Por ejemplo, la reciprocidad a la que antes se aludía, se presenta en los marcos que verifican

$$\forall x \forall y (R_a xy \rightarrow R_b yx)$$

Resulta, sin embargo, mucho más útil si la dependencia entre relaciones se puede caracterizar mediante una o varias fórmulas modales. Por ejemplo, en todos los marcos en que son válidas las fórmulas

$$p \rightarrow [a] \langle b \rangle p \qquad\qquad p \rightarrow [b] \langle a \rangle p$$

las relaciones R_a y R_b son recíprocas.

Aplicaciones temporales Tal como se decía al inicio de esta sección, aplicaciones de interés requieren de lenguajes con más de un operador modal. La lógica temporal, objeto de estudio en esta tesis, es un tipo de lógica polimodal.

2.5. Lógica temporal

Las lógicas de predicados y clásica se encuentran limitadas para expresar razonamientos que incluyan modalidades como la temporalidad, dado que se basan en declaraciones y proposiciones que son válidas de manera indefinida. No toda proposición debe ser válida para siempre, ni desde siempre. Así, algunas veces, surge

una necesidad de incluir temporalidad, con lo cual, nace la lógica temporal que busca especificar, expresar y razonar comportamientos dinámicos (Torres, 2007).

En la lógica clásica, una proposición es verdadera o es falsa (Torres, 2007); por ejemplo: todos los hombres son de Marte, María está en el bosque, $X = 5$, etc. Sin embargo, las valoraciones "verdadero" o "falso" se pueden considerar dependientes del instante (tiempo) y el espacio (sitio) cuando se consideran expresiones como: "el cielo está claro", "el parqueo se encuentra lleno", "$X = 10$ y $X = 5$". Es decir, en estas proposiciones el valor de veracidad dependerá del tiempo.

De manera particular, la proposición "$X = 10$ y $X = 5$" es falsa en lógica clásica, pero, la proposición, podría ser cierta si la variable X es dependiente del tiempo, con lo cual se tendría que utilizar una fórmula tal como $\exists t, t'$ tal que $[t < t'] \wedge [(X(t) = 10 \wedge X(t') = 5)]$. Por ende, se estaría especificando un comportamiento dinámico (Torres, 2007).

Formalmente, la Lógica Temporal es una extensión del cálculo de predicados en la cual se incluyen ciertos operadores lógicos, modales y dependientes del tiempo (Torres, 2007). Así, el término Lógica Temporal ha sido ampliamente utilizado para cubrir todos los enfoques de representación de información relacionada con el tiempo dentro de un marco lógico. Más específicamente se ha utilizado este término para referirse a la lógica de tipo modal introducida en los años 60 por Arthur Prior bajo el nombre de Lógica del Tiempo y que fue luego desarrollada por lógicos y científicos de la computación (Galton, 2008).

Las aplicaciones de la Lógica Temporal incluyen (Galton, 2008) su uso para la clarificación de cuestiones filosóficas acerca del tiempo, como un marco de trabajo dentro del cual se define la semántica de las expresiones temporales en lenguaje natural, como un lenguaje para codificar información temporal en inteligencia arti-

ficial y como una herramienta para manejar los aspectos temporales de la ejecución de programas de computadora.

2.5.1. Lógica temporal básica

2.5.1.1. Sintaxis y semántica

La lógica temporal básica es una lógica modal proposicional con dos operadores: $\langle P \rangle$ y $\langle F \rangle$. De cada uno de ellos es útil considerar su respectivo operador dual e inclusive reservarle un símbolo específico[9]:

$$[H] = \neg \langle P \rangle \neg \qquad [G] = \neg \langle F \rangle \neg$$

La lectura pretendida para $\langle P \rangle$ y $\langle F \rangle$ es:

$\langle P \rangle \phi$ existe al menos un mundo (un instante) en el pasado en el cual se satisface ϕ

$\langle F \rangle \phi$ existe al menos un mundo (un instante) en el futuro en el cual se satisface ϕ

Con esta lectura, sus operadores duales deben interpretarse como:

$[H\phi] = \neg \langle P \rangle \neg\phi$ No existe un instante en el pasado en que no se satisfaga ϕ en todo instante pasado se satisface ϕ.

$[G\phi] = \neg \langle F \rangle \neg\phi$ No existe un instante en el futuro en que no se satisfaga ϕ en todo instante futuro se satisface ϕ.

[9] **Notación:** se han respetado las iniciales usuales, que en inglés corresponden con:

- $\langle P \rangle$: para algún instante en el pasado (*Past*)
- $\langle F \rangle$: para algún instante en el futuro (*Future*)
- $[H]$: para todo instante pasado (*it always Has been ...*)
- $[G]$: para todo instante futuro (*it always is Going to ...*)

Usualmente P, F, H y G se escriben sin corchetes. Aquí los corchetes se han añadido para recalcar el carácter existencial de P y F frente al universal de H y G, sus respectivos operadores duales.

2.5.1.2. Modelización adecuada del tiempo

El lenguaje temporal básico se puede interpretar en cualquier marco que facilite dos relaciones binarias R_P y R_F (Fernández et al. 2007).

Transitividad Parece razonable que, si en el futuro de w_1 se encuentra w_2 y en el futuro de w_2 se encuentra w_3, entonces en el futuro de w_1 debiera estar w_3. Es decir, que la relación R_F sea transitiva:

$$\forall w_i w_j w_k (R_F w_i w_j \wedge R_F w_j w_k \rightarrow R_F w_i w_k)$$

La misma transitividad se puede pedir de R_P, que relaciona a un instante w_i con otro w_j en su pasado: $R_P w_i w_j$. Así, como mínimo, no se interpreta la lógica temporal básica sobre cualquier marco $\langle W, R_P, R_F \rangle$, sino sobre aquéllos donde tanto R_P como R_F sean transitivas. Es decir, en los marcos en que sean válidas las fórmulas:

$$[H]\phi \rightarrow [H][H]\phi \qquad [G]\phi \rightarrow [G][G]\phi$$

o sus equivalentes

$$\langle P \rangle \langle P \rangle \psi \rightarrow \langle P \rangle \psi \qquad \langle F \rangle \langle F \rangle \psi \rightarrow \langle F \rangle \psi$$

Reciprocidad pasado-futuro Otra restricción razonable introduce una dependencia entre ambas relaciones: se quisiera que en el futuro de w_1 se encuentre w_2 si y sólo si en el pasado de w_2 se encuentra w_1. Es decir, que una relación sea recíproca de la otra. Se mencionó que los marcos donde esto se verifica son aquellos en que son válidas las dos fórmulas siguientes:

$$p \rightarrow [H] \langle F \rangle p \qquad p \rightarrow [G] \langle P \rangle p$$

Cuando se verifica esta reciprocidad, se puede considerar que existe una única relación R (por ejemplo, $R = R_F$). Entonces, la relación 'hacia el pasado' utiliza R pero hacia atrás. Formalmente, basta redefinir la semántica de $\langle P \rangle$ y $[H]$:

$M, w \Vdash \langle F \rangle \phi$ si y sólo si existe un $w' \in W$ tal que Rww' y $M, w' \Vdash \phi$

$M, w \Vdash [G]\phi$ si y sólo si para todo $w' \in W$, si Rww' entonces $M, w' \Vdash \phi$

$M, w \Vdash \langle P \rangle \phi$ si y sólo si existe un $w' \in W$ tal que $R\mathbf{w}'\mathbf{w}$ y $M, w' \Vdash \phi$

$M, w \Vdash [H]\phi$ si y sólo si para todo $w' \in W$, si $R\mathbf{w}'\mathbf{w}$ entonces $M, w' \Vdash \phi$

Obsérvese cómo, en la semántica de $\langle P \rangle$ y $[H]$, los instantes se relacionan por R de forma inversa a cómo lo hacen para $\langle F \rangle$ y $[G]$.

Otras restricciones opcionales Añadiendo restricciones adicionales se pueden ir consiguiendo unas u otras modelizaciones más específicas. Por ejemplo, si se exige que todo instante tenga uno y sólo un R-sucesor, nos limitamos a los marcos temporales lineales; si se permiten varios R-sucesores, consideramos líneas de tiempo que se bifurcan. Asimismo se pueden considerar modelos discretos o modelos densos.

De manera más formal, estas restricciones pueden ser (Venema, 2001):

Tener un punto de inicio	$[H]\bot \vee \langle P \rangle [H]\bot$	(A1)
Serialidad hacia la izquierda	$\langle P \rangle \top$	(A2)
Tener un punto final	$[G]\bot \vee \langle F \rangle [G]\bot$	(A3)
Serialidad hacia la derecha	$\langle F \rangle \top$	(A4)
Ser discreto	$(\langle F \rangle \top \wedge q \wedge [H]q) \rightarrow \langle F \rangle [H]q$	(A5)
Densidad	$\langle F \rangle q \rightarrow \langle F \rangle \langle F \rangle q$	(A6)
Continuidad	$(\langle F \rangle q \wedge \Diamond\neg q \wedge \Box(q \rightarrow [H]q)) \rightarrow$ $\Diamond((q \wedge \neg[G]q) \vee (\neg q \wedge [H]q))$	(A7)
Tener intervalos finitos	$([G]([G]q \rightarrow q) \rightarrow (\langle F \rangle [G]q \rightarrow$ $[G]q)) \wedge ([H]([H]q \rightarrow q) \rightarrow$ $(\langle P \rangle [H]q \rightarrow [H]q))$	(A8)

2.5.1.3. Lógica temporal mínima

Existe una lógica *temporal mínima* llamada $\mathbf{K_t}$.[10]

Definición 2.5.1. (Lógica temporal mínima K_t) Esta lógica se define como la clase más pequeña de fórmulas del tiempo, que es cerrada bajo los siguientes axiomas y reglas de derivación (Venema, 2001):

(CT)	todas las proposiciones tautológicas clásicas	
(DB)	$[G](q \to r) \to ([G]q \to [G]r)$ $[H](q \to r) \to ([H]q \to [H]r)$	(*Distributividad*)
(CV)	$q \to [G]\langle P\rangle q$ $q \to [H]\langle F\rangle q$	(*Conversión*)
(4)	$[G]q \to [G][G]q$	(*Transitividad*)
(US)	si ϕ es un teorema, entonces también lo es ϕ^σ (ver la definición 2.4.4)	(*Sustitución uniforme*)
(MP)	si ϕ y $\phi \to \psi$ son teoremas, entonces también lo es ψ	(*Modus ponens*)
(TG)	si ϕ es un teorema, entonces también lo son $[G]\phi$ y $[H]\phi$	(*Generalización temporal*)

Definición 2.5.2. (Lógica Lin) La lógica **Lin** que es la clase de flujos lineales del tiempo es una extensión de $\mathbf{K_t}$ por medio del axioma (NB). (NB) es la conjunción

[10]Como nota anecdótica, esta lógica es análoga a la lógica modal normal mínima **K** (definición A.3.1 del apéndice).

del axioma $\langle P\rangle \langle F\rangle q \rightarrow (\langle P\rangle q \vee q \vee \langle F\rangle q)$ (que prohibe la bifurcación en el futuro) y su recíproco hacia el pasado $\langle F\rangle \langle P\rangle q \rightarrow (\langle F\rangle q \vee q \vee \langle P\rangle q)$ (Venema, 2001).

Definición 2.5.3. (Lógicas Lin.N, Lin.Z, Lin.Q, Lin.R) Las axiomáticas de estas estructuras específicas se definen así (Venema, 2001):

Lin.N: Lin + A1 + A4 + A8

Lin.Z: Lin + A2 + A4 + A8

Lin.Q: Lin + A2 + A4 + A6

Lin.R: Lin + A2 + A4 + A6 + A7

Para estas lógicas aplica el siguiente resultado:

Teorema 2.5.4. *Las lógicas **Lin.N**, **Lin.Z**, **Lin.Q** y **Lin.R** son axiomatizaciones sólidas y completas de los conjuntos de validez de los flujos de tiempo $\mathcal{N}$, $\mathcal{Z}$, $\mathcal{Q}$ y $\mathcal{R}$, respectivamente (Venema, 2001).*

Este resultado es útil en la programación lógica temporal que se explica en seguida.

2.6. Patrones secuenciales en lógica temporal

En la sección 2.3 se mencionó que la lógica temporal permite expresar orden o secuencias, con lo cual, la definición de dato secuencial es toda secuencia expresable por medio de fórmulas de lógica del tiempo. Sin embargo, la clase de todas las secuencias expresables por medio de fórmulas de lógica del tiempo es infinita. Así, únicamente se estudia un tipo de secuencia. Considérense los patrones secuenciales, definidos por Dunham (2002) y mencionados en la sección

2.3 de la forma $S = \langle s_1, s_2, \ldots, s_n \rangle$ (donde $s_i \subseteq I$ e I es un conjunto de artículos o elementos, $I = \{I_1, I_2, \ldots, I_m\}$). Este patrón ha sido muy estudiado desde Agrawal y Srikant (1994b). Se considera a este patrón frecuente si existe cierto porcentaje α de clientes que compran el conjunto de artículos s_j secuencialmente. O sea, los artículos en s_1 son comprados en el tiempo t_1, los artículos en s_2 son comprados en el tiempo t_2, y así, donde $t_1 < t_2 < \ldots < t_n$. Si se denota como $p_k^j (k = 1, \ldots, n)$ a las variables proposicionales que representan los artículos en el conjunto s_j, entonces este patrón puede ser expresado en lógica temporal proposicional por la fórmula $s_1 \wedge \langle F \rangle (s_2 \wedge \langle F \rangle (s_3 \wedge \langle F \rangle (\ldots \wedge \langle F \rangle s_n) \ldots))$, donde s_j es la fórmula $(p_1^j \wedge p_2^j \wedge \ldots p_{n_j}^j)$ (De Amo, Junior et al. 2007). Por supuesto, podemos hacer generalizaciones de este tipo de patrón como, por ejemplo, expresar secuencias de lógica temporal de primer orden. Así, el patrón "*El cliente renta* La guerra de las galaxias*, luego* El imperio contraataca *y luego* El regreso del Jedi" en ese orden, se podría expresar:

$$\begin{aligned} &\text{ClienteRenta}(\text{"La guerra de las galaxias"}) \quad \wedge \\ &\langle F \rangle \, (\text{ClienteRenta}(\text{"El imperio contraataca"}) \quad \wedge \\ &\langle F \rangle \, \text{ClienteRenta}(\text{"El regreso del Jedi"})) \end{aligned}$$

En abstracto, el patrón sujeto de estudio tiene la forma $x_1 \wedge \langle F \rangle (x_2 \wedge \langle F \rangle (x_3 \wedge \langle F \rangle (\ldots \wedge \langle F \rangle x_n) \ldots))$. La siguiente sección describe brevemente cómo es posible plantear lógica temporal en programación lógica.

2.7. Programación lógica temporal

Un programa de programación lógica temporal consiste en un conjunto de reglas temporales de la forma CABEZA←CUERPO, donde diferentes sistemas de programación lógica temporal hacen suposiciones diferentes acerca de la estructu-

ra de CABEZA y CUERPO (Padmanabhan & Tuzhilin, 1996). Por ejemplo, una regla tal como "los empleados que han sido despedidos de una empresa no pueden ser recontratados por esa empresa en el futuro" puede ser expresada en un lenguaje de programación lógica temporal llamado *Templog* así:

$$\neg\text{CONTRATA(empresa, persona)}$$
$$\leftarrow \langle P \rangle \text{ CONTRATA(empresa, persona)} \wedge \neg\text{CONTRATA(empresa, persona)}$$

De manera alternativa, tal como se mencionará luego, es posible expresar programas de programación lógica temporal en lógica de primer orden utilizando referencias de tiempo explícitas pues esto es semánticamente equivalente. Por ejemplo, en lugar de utilizar el predicado temporal CONTRATA(empresa, persona), podemos utilizar su equivalente en lógica de primer orden CONTRATA(empresa, persona, tiempo) especificando el historial de empleo de la persona en el tiempo. Aún más, *Templog* y el lenguaje correspondiente de primer orden son equivalentes en poder expresivo.

Ejemplo 2.7.1. Un ejemplo de una consulta temporal inductiva que podría haber sido planteada por un epidemiólogo (Chomicki, 1994) es:

> Encuentre todas las personas en situación de riesgo donde "estar en riesgo" se define de la siguiente manera: una persona "está en riesgo" en un momento dado si ha sido infectado antes o ha estado en contacto con alguien que ya está en riesgo.

Considérese que haber estado en contacto con alguien que se convirtió a "en riesgo" posteriormente, no debe resultar en la clasificación de la persona como en situación de "en riesgo". Debe quedar claro que la consulta anterior no es de primer-orden debido a la recursividad inherente.

Las consultas inductivas temporales se pueden formular en una serie de lenguajes de programación lógica. Estos lenguajes extienden de varias maneras a *Datalog*, el lenguaje de programación lógica libre de funciones. Considérense los siguientes lenguajes:

- $Datalog^{<\mathcal{Z}}$: *Datalog* con restricciones de orden de números enteros.
- $Datalog^{<\mathcal{Q}}$: *Datalog* con restricciones de orden de números racionales.
- $Datalog_{1S}$: *Datalog* con un símbolo sucesor unario en un argumento.
- *Datalog* con un símbolo sucesor unario y restricciones lineales aritméticas.

Estos lenguajes son un subconjunto de *Prolog*, así que su implementación únicamente requiere de una adaptación de las técnicas de programación lógica existentes. $Datalog_{1S}$ es aplicable únicamente a estructuras temporales finitas, mientras que los demás lenguajes son aplicables a bases de datos concretas de marcas de tiempo.

La formulación en $Datalog^{<\mathcal{Z}}$ (o $Datalog^{<\mathcal{Q}}$) de la consulta del ejemplo 2.7.1 es muy natural:

$$\begin{aligned} enRiesgo(X,T) &\leftarrow infectado(X,T'), T' < T. \\ enRiesgo(X,T) &\leftarrow contacto(X,Y,T'), enRiesgo(Y,T'), T' < T. \end{aligned}$$

La formulación en $Datalog_{1S}$ es menos transparente, porque el símbolo de relación de orden < no está directamente disponible:

$$\begin{aligned} enRiesgo(X,T+1) &\leftarrow infectado(X,T) \\ enRiesgo(X,T+1) &\leftarrow enRiesgo(X,T) \\ enRiesgo(X,T+1) &\leftarrow contacto(X,Y,T), enRiesgo(X,T). \end{aligned}$$

Debido a las restricciones sintácticas en el número de argumentos en los que el símbolo de sucesor puede aparecer, $Datalog_{1S}$ no puede expresar algunas consultas $Datalog^{<\mathcal{Z}}$.

Los lenguajes deductivos mencionados se pueden ampliar con negación estratificada en el cuerpo de las cláusulas. Tal extensión de $Datalog_{1S}$ se denota $Datalog_{1S}{}^{\neg}$ *estratificado*.

Ejemplo 2.7.2. La consulta del ejemplo 2.7.1 se pueden expresar en $Datalog^{<\mathcal{Z}}$ con negación estratificada como:

$$\begin{aligned} algunI(X,T) &\leftarrow I(X,S,T). \\ consulta(X) &\leftarrow I(X,S_1,T_1), I(X,S_2,T_2), \neg algunI(X,T), T_1 < T < T_2 \end{aligned}$$

Así, definida la programación lógica temporal, se discute en el capítulo siguiente el modelo para la minería de patrones secuenciales.

Capítulo 3

Modelo para la minería de patrones secuenciales

Este capítulo introduce un modelo diseñado para la minería de patrones secuenciales que incluye una etapa de conteo de fórmulas de lógica temporal. Primero se describe el modelo, luego se dan detalles sobre su implementación.

3.1. Descripción del modelo

El *modelo de minería de patrones secuenciales utilizando lógica temporal* consiste de lo siguiente:

1. Una etapa de transformación donde se considera una base de datos relacional como una base de datos temporal.

2. Otra etapa de transformación donde se considera la base de datos temporal como una base de datos temporal deductiva.

3. Un algoritmo de detección de patrones secuenciales que consiste de cuatro fases: una subetapa de transformación de la relación a analizar[1] en una base de datos de secuencias, una búsqueda de todos los patrones de tamaño 1 que servirán como semilla de la detección de patrones, la detección de los patrones secuenciales en la base de datos de secuencias y, por último, la eliminación de los patrones no maximales, es decir, los patrones contenidos en otros.

En la sección 1.2 se mencionó que existen varias etapas previas a la minería de datos. Estas etapas se pueden dar de diferentes maneras, de manera unificada y de manera opcional. Así, para las etapas de limpieza de los datos, integración de los datos, selección de los datos y transformación de los datos, esta investigación propuso una vista de base de datos temporal abstracta deductiva, la cual fue tomada como objetivo de transformación. El modelo realizado primero transforma los datos a una vista de base de datos temporal lo cual se discute en la sección siguiente.

3.1.1. Base de datos temporal abstracta: vista de teoría de modelos

La vista de base de datos temporal abstracta, que se define a continuación, es idéntica a la definida en Chomicki (1994) por ser esta suficiente. Se asume que existe una única dimensión de tiempo, un único dominio de tiempo $\mathcal{T}$ y un único dominio de datos $\mathcal{U}$. Estas suposiciones se discuten dentro del contexto de modelos de datos relacionales pero pueden ser generalizables a otros modelos de datos. Además, se asume que el esquema de la base de datos es fijo y que consiste de un conjunto fijo de relaciones.

Para esta vista considérese una relación $\mathcal{P}$ de aridad n. Con el fin de modelar el hecho de que las tuplas de $\mathcal{P}$ corresponden a los hechos que se satisfacen en un

[1] Tabla del modelo relacional.

instante de tiempo, se presenta la *relación temporal abstracta* $\mathcal{P}'$ de aridad n + 1 cuyas tuplas tienen el siguiente significado:
$(t, a_1, \ldots, a_n) \in \mathcal{P}'$ sí y sólo sí $P(t, a_1, \ldots, a_n)$ se satisface en el instante t donde $a_1, \ldots, a_n \in \mathcal{U}$.

De manera más formal, es bien conocido que una base de datos relacional puede ser vista como una estructura finita $D = (\mathcal{U}, P_1, \ldots, P_k)$ para el lenguaje de primer orden L_D que contiene símbolos de relación para todas las relaciones $P_1, \ldots, P_k$ en la base de datos y símbolos de constante para todos los elementos de $\mathcal{U}$. Una *base de datos temporal abstracta* correspondiente (llamada una *estructura temporal*) es una estructura $D' = (\mathcal{U}, \mathcal{T}, P'_1, \ldots, P'_k)$ para el lenguaje de primer orden y doblemente ordenado L'_D que contiene un símbolo de relación temporal nuevo para cada relación temporal abstracta P'_i y símbolos de constante para al menos todos los elementos de $\mathcal{U}$ (y posiblemente algunos elementos de $\mathcal{T}$ también). La aridad de P'_i es la aridad de $P_i + 1$. El argumento extra de P'_i (primero por convención) es llamado *temporal*, otros argumentos son llamados *datos*. El lenguaje L_D contiene también la relación y símbolos de función, como por ejemplo, "$<$" o "$+$", del lenguaje de dominio temporal $\mathcal{T}$ (estos símbolos no son interpretados en D' pero tienen un significado fijo).

Nótese que el concepto de base de datos abstracta temporal está parametrizado por el dominio temporal $\mathcal{T}$ y no se hace ninguna suposición sobre $\mathcal{T}$. Se dice que una base de datos D' es finita si consiste de relaciones finitas. Así, la definición arriba permite bases de datos finitas e infinitas.

Dicho de manera informal, para transformar una base de datos relacional a base de datos temporal es suficiente escoger alguna columna (a_i según la definición anterior) como el parámetro temporal (t según la definición anterior), siempre y cuando se puedan aplicar posteriormente los símbolos de función del lenguaje L_D provenientes del lenguaje de dominio temporal $\mathcal{T}$.

3.1.2. Base de datos temporal deductiva

Un sistema de base de datos deductiva es un sistema de base de datos que puede hacer deducciones (concluir hechos adicionales) basándose en las reglas y los hechos almacenados en la base de datos. *Datalog* es usualmente el lenguaje utilizado para especificar estos hechos, reglas y consultas en bases de datos deductivas.

Las bases de datos deductivas nacieron del deseo de combinar la programación lógica con las bases de datos relacionales para construir sistemas que apoyaran un formalismo poderoso y que fueran rápidas y capaces de lidiar con grandes conjuntos de datos. Las bases de datos deductivas son más expresivas que las bases de datos relacionales, aunque menos expresivas que los sistemas de programación lógica. Inclusive la traducción de *Datalog* a *SQL* parece ser bastante simple (Dehaspe & Toivonen, 1999). Por ejemplo, la consulta Datalog:

?− window(Window_id, Alarmtype), is_a(Alarmtype, warning)

Se traduce a SQL así:

```
SELECT WINDOW.window_id, WINDOW.alarmtype
FROM WINDOW, IS_A
WHERE WINDOW.alarmtype=IS_A.alarmtype AND IS_A.ancestor='warning'
```

De manera más formal (Das, 1992), una *base de datos deductiva* es un par $(D, \mathcal{L})$, donde D es un conjunto finito de cláusulas de bases de datos y $\mathcal{L}$ es un lenguaje de primer orden. Se asume que $\mathcal{L}$ tiene al menos dos símbolos, uno que representa un símbolo de constante y otro que representa un símbolo de predicado. Una *base de datos relacional* es una base de datos deductiva $(D, \mathcal{L})$ donde D contiene únicamente hechos definidos (cláusulas básicas[2] en términos de programación lógica). Por lo tanto, una base de datos relacional es una forma especial de base de datos deductiva.

[2]Es común encontrar el término cláusula *ground* en lugar de cláusula básica.

Así, dado que la estructura de cláusulas de bases de datos y la estructura de cláusulas de programas lógicos son iguales (Das, 1992), se puede simular una base de datos deductiva a través de un sistema de programación lógica, montando los hechos de una base de datos relacional como cláusulas básicas del sistema y agregando traducción del lenguaje de programación lógica hacia *SQL* para que la utilización de las cláusulas básicas sea transparente en el sistema.

Por último, si se unen el concepto de base de datos deductiva y el concepto de base de datos temporal anterior se obtiene un sistema de base de datos temporal deductiva. Esto lleva el objetivo de transformación de los datos iniciales al punto requerido para aplicar un algoritmo inductivo de detección de patrones secuenciales en D'.

3.1.3. Algoritmo de detección de patrones secuenciales

Algoritmo 3.1: Algoritmo para encontrar todos los patrones secuenciales frecuentes y maximales de la forma $x_1 \wedge \langle F \rangle\ (x_2 \wedge \langle F \rangle\ (x_3 \wedge \langle F \rangle\ (\ldots \wedge \langle F \rangle\ x_n)\ldots))$

Entrada: Base de datos D' y umbral de apoyo n

Salida : Patrones de tamaño k con apoyo n en la base de datos

1. Transformación de la base de datos de eventos (transacciones) D' a una base de datos $\mathcal{S}$ de secuencias
   (refiérase a la sección 3.1.3.1)
2. Búsqueda en $\mathcal{S}$ de toda la lista de conjuntos de elementos grandes (L={conjuntos grandes tamaño 1}) con apoyo n de la base de datos
   (refiérase a la sección 3.1.3.2)
3. Búsqueda de patrones secuenciales en $\mathcal{S}$
   (refiérase a la sección 3.1.3.3)
4. Eliminación de patrones no maximales

Se puede notar que los pasos principales son de manera general los mismos pasos que utilizados por Agrawal y Srikant (1994; 1995). Efectivamente, este es el mismo esquema utilizado por una serie de algoritmos en la misma área de investigación. Sin embargo, en el paso 3 no se utiliza ninguno de los algoritmos AprioriAll o AprioriSome sino que se hace un conteo de reglas temporales de la forma $x_1 \wedge \langle F \rangle\ (x_2 \wedge \langle F \rangle\ (x_3 \wedge \langle F \rangle\ (\ldots \wedge \langle F \rangle\ x_n)\ldots))$ sobre la base de datos de secuencias $\mathcal{S}$. Además, la fase o paso 3 hace una búsqueda en profundidad primero a diferencia de una gran mayoría de algoritmos que utilizan la búsqueda en ancho primero para encontrar los patrones, v.gr. (Agrawal & Srikant, 1994b). La búsqueda en profundidad primero permite aprovechar las características de la programación lógica.

3.1.3.1. Transformación de la base de datos de eventos a una base de datos de secuencias

Este paso del algoritmo consiste en hacer la misma transformación realizada por Agrawal y Srikant (1994b). Sin embargo, en este modelo hay una adaptación importante: cada conjunto de elementos de cada secuencia tiene un parámetro temporal. En otras palabras, se registra el tiempo en que ocurre cada evento, mientras que en el trabajo de Agrawal y Srikant (1994b) no se hace esto, sino que se tienen los conjuntos de elementos uno tras otro en el orden que ocurrieron pero sin parámetro temporal. La importancia de esta diferencia radica en que ese parámetro temporal es el que permite la utilización de un lenguaje lógico con equivalencia semántica a la lógica temporal.

Por otro lado, es importante hacer notar que, a diferencia de Agrawal y Srikant (1994b), no se realizó la asociación de conjuntos de elementos a identificadores.

3.1.3.2. Generación de la lista de conjuntos de elementos grandes

El apoyo en los datos a un conjunto de elementos se define como la fracción de secuencias en $\mathcal{S}$ que contienen al conjunto de elementos. Así, el conjunto de elementos i y el patrón secuencial de tamaño $1 < i >$ tienen el mismo apoyo en los datos[3]. Un conjunto de elementos con apoyo mínimo de los datos es llamado un conjunto de elementos grande o *litemset*. Esta fase del algoritmo consiste, entonces, en encontrar el conjunto de todos los conjuntos de elementos grandes L, lo cual es equivalente a encontrar los patrones secuenciales de tamaño 1.

El problema de encontrar conjuntos de elementos grandes dado un conjunto de transacciones, aunque con una definición de apoyo en los datos un tanto diferente,

[3]Los patrones secuenciales de tamaño 1 no son interesantes como resultado pues no implican relación temporal. Sin embargo, son interesantes en el sentido de que brindan un punto de partida para patrones de tamaño mínimo 2 que sí poseen una característica temporal.

fue considerado en Agrawal, Imielinski y Swami (1993), Houtsma y Swami (1993), Agrawal y Srikant (1994a), Mannila, Toivonen y Verkamo (1994) antes del trabajo sobre patrones secuenciales de Agrawal y Srikant (1994b). Existe investigación posterior a Agrawal y Srikant (1994b) sobre el mismo tema tal como en Park, Chen y Yu (1995) y otros. Inclusive existe investigación más reciente como Han, Pei y Yin (2000). Sin embargo, en estas investigaciones el apoyo de los datos del conjunto de elementos ha sido definido como la fracción de transacciones en las cuales el conjunto de elementos está presente, mientras que en el problema de encontrar patrones secuenciales frecuentes el apoyo es la fracción de secuencias en $\mathcal{S}$ que contienen al menos una vez el conjunto de elementos. Los algoritmos de las investigaciones mencionadas son fácilmente modificables para considerar esta diferencia (Agrawal & Srikant, 1994b). Así, en esta fase del algoritmo, tal como se comenta en Agrawal y Srikant (1994b), se podría utilizar APRIORITID o APRIORIHYBRID (la combinación de APRIORI y APRIORITID) o cualquier otro algoritmo de las investigaciones mencionadas.

La generación de la lista de conjuntos de elementos grandes se hace de la misma manera que en Agrawal y Srikant (1994b). Se utiliza APRIORI simple para esta generación. Sin embargo, tal como se comenta en Agrawal y Srikant (1994b) se hace una modificación principal a APRIORI la cual es que el conteo de apoyo debe ser incrementado únicamente una vez por cada secuencia en $\mathcal{S}$ que contiene al conjunto de elementos aunque el conjunto de elementos esté presente varias veces.

Por supuesto, existen técnicas más modernas aplicables a este paso como árboles de patrones frecuentes (Han, Pei et al. 2000). Inclusive, en este paso del algoritmo, se puede profundizar en la técnica, por ejemplo con paralelización. Dado que el objetivo principal de esta tesis es el estudio de la lógica temporal y su aplicabilidad en el descubrimiento de patrones temporales, no se profundizó aún más en el estudio

de esta fase del algoritmo. Lo importante fue obtener el conjunto L de conjuntos de elementos grandes para la siguiente etapa que se explica en seguida y donde se hace uso de lógica temporal.

3.1.3.3. Búsqueda de los patrones utilizando lógica temporal

Algoritmo 3.2: Búsqueda de los patrones utilizando lógica temporal

Entrada: Base de datos $\mathcal{S}$, lista L de conjuntos de elementos grandes y umbral de apoyo n

Salida : Conjunto P de patrones de tamaño k con apoyo n de la base de datos

$P \leftarrow \varnothing$;

para cada $l \in L$ **haga**

```
/* Se generan patrones utilizando l como raíz llamando a
   generar(l, L, S, n, P) (véase el algoritmo 3.3)          */
```

generar(l,L, $\mathcal{S}$, n, P);

fin

devolver P;

El algoritmo 3.2 únicamente toma cada elemento en L como raíz o patrón de tamaño 1 y llama a generar(l, L, $\mathcal{S}$, n) para buscar recursivamente y en profundidad primero apoyo de los datos (por medio de conteo) a patrones secuenciales. Esa búsqueda se hace tal como en el algoritmo a continuación.

Algoritmo 3.3: Generación de patrones: generar(l_k, L, $\mathcal{S}$, n, P)

Entrada: Fórmula temporal l_k, lista L de conjuntos de elementos grandes, base de datos $\mathcal{S}$, umbral de apoyo n y conjunto P de patrones

para cada $l \in L$ **haga**

`/* El apoyo es un conteo de la aparición de la fórmula` $l_k \wedge \langle F \rangle\, l$ `en` $\mathcal{S}$`, la aparición se cuenta una vez por cada secuencia de` $\mathcal{S}$ `donde la fórmula sea verdadera (ver la definición 2.4.11) */`

si *apoyo de* $l_k \wedge \langle F \rangle\, l$ *en* $\mathcal{S} \geq n$ **entonces**

$P \leftarrow P \cup l_k \wedge \langle F \rangle\, l$;

generar($l_k \wedge \langle F \rangle\, l$, L, $\mathcal{S}$, n, P);

fin

fin

3.2. Implementación del modelo

En esta sección se discute la implementación que se realizó del modelo. Se explica su arquitectura de componentes y se dan algunos detalles sobre la implementación. Es posible encontrar la implementación completa del modelo a través de su código fuente en el sistema de control de versiones en la dirección `https://sourceforge.net/projects/chronos-tmw/`. A la aplicación resultante de esta implementación se le ha llamado "Chronos: Temporal Mining Workbench", en adelante *chronos-tmw*.

3.2.1. Base de datos relacional como base de datos temporal deductiva

Para realizar un sistema de minería de patrones secuenciales, en bases de datos, utilizando lógica temporal es necesario tener programación lógica en bases de datos. Dado que el objetivo de esta investigación no era el de producir herramientas de consulta en bases de datos, se buscó reutilizar un sistema ya implementado de *Datalog* o de programación lógica que además fuese libre. El cuadro 3.2.1 muestra un resumen de los sistemas de programación lógica en bases de datos encontrados.

Cuadro 3.2.1: Implementaciones libres de Datalog.

Implementación Datalog	Detalles
BDD-Based Deductive DataBase (bddbdd)	Es una implementación de Datalog realizada en Stanford. Está disponible en `http://bddbddb.sourceforge.net/`.
Datalog	Es un sistema liviano de base de datos deductiva implementado en el lenguaje de programación Lua. Se encuentra disponible en `http://www.ccs.neu.edu/home/ramsdell/tools/datalog/`.
Datalog Educational System (DES)	Es una implementación de base de datos deductiva basada en Prolog. Es posible encontrarla en `http://www.fdi.ucm.es/profesor/fernan/DES/`.
Coral	Es un sistema de bases de datos deductivas. Se puede encontrar en `http://www.cs.wisc.edu/coral/`. Ventaja: posee una interfaz C++ lo que facilita tanto la programación declarativa como la imperativa. Desventaja: es "viejo", es una implementación de 1997.
LDL++	Es un sistema de base de datos deductiva disponible en `http://www.cs.ucla.edu/ldl/`. Ventajas: tiene capacidad de conexión con sistemas de gestión de bases de datos. Es muy poderoso pues tiene muchas características: minería de datos, hilog, no-monotónico, etc. Desventaja: sólo es libre para usos no comerciales.

Implementación Datalog	Detalles
XSB	Es un sistema de base de datos deductiva y programación lógica. Se encuentra disponible en `http://xsb.sourceforge.net/`. Ventajas: tiene una buena base de características: interfaz C, interfaz ODBC (traduce Datalog a SQL), interfaz Java, interfaz hacia Perl (RegEx), HiLog, etc. Su licencia es libre (GPL).

Tomando en cuenta que *XSB:*

- provee un ambiente de programación lógica que permite a los usuarios declarar predicados *XSB* que conectan a tablas individuales en una base de datos conectada al ambiente por medio de algún controlador,
- ha mostrado ser un sistema de base de datos deductiva eficiente (Sagonas, Swift & Warren, 1994),
- es un sistema con licenciamiento libre,
- es mejorado continuamente,
- puede ser empotrado en programación imperativa *Java* (Declarativa - Serviços de Informática, Lda. 2011) y
- está bien documentado (Swift et al. 2011; XSB Group, 2011a),

se tomó la decisión de utilizar este sistema.

A pesar de utilizar *XSB*, no se aprovecharon dos características suyas que lo hacen muy atractivo: resolución tabulada y programación lógica de alto orden. Se consideró utilizar estas dos características pues hacen la programación más sencilla. Sin embargo, se tomó la decisión de no hacerlo para que la programación lógica

fuese fácilmente portable a *SWI-Prolog* (SWI-Prolog, 2011) dado que este sistema tiene un mejor ambiente y posee depuración gráfica. Así, algunos predicados se desarrollaron en paralelo en ambos sistemas únicamente para poder ser depurados de manera más amigable. Además, el hecho de que el predicado `time/1` sea más informativo en *SWI-Prolog* que en *XSB* también influyó. Efectivamente, el código de programación lógica *XSB* final en *chronos-tmw* es fácilmente portable a *SWI-Prolog* luego de pocos cambios.

Otra decisión de implementación tomada fue la de no restringirse a *Datalog*. *Datalog* en *XSB* es perfectamente simulable eliminando los operadores `\+/1`, `fail`, `if/1`, `not/1`, `tnot/1`, `setof/3`, `bagof/3`, `findall/3` u otros operadores de sumarización y utilizando resolución tabulada sobre los predicados de interés, ya que ésta garantiza una propiedad de profundidad acotada de los términos y, por lo tanto, la terminación de la computación (Swift et al. 2011). Así, la implementación fue un tanto más compleja dado que para garantizar la terminación fue necesario atenerse a la semántica procedimental de *Prolog*, donde el orden de los predicados tiene importancia y permite, únicamente a través de la responsabilidad del programador, garantizar la terminación.

Finalmente, para tener una base de datos relacional como base de datos temporal deductiva se combina la vista de base de datos temporal mencionada en la sección 3.1.1 junto con la capacidad de *XSB* de conectar a tablas individuales en bases de datos mencionada en abstracto en la sección 3.1.2. Además, tal como se indica en Padmanabhan y Tuzhilin (1996), Chomicki (1994) no es necesario tener un lenguaje de programación lógica con operadores de lógica temporal, sino que es suficiente con tener un argumento *temporal*, así como se define en la secciones 2.7 y 3.1.1 pues esto es semánticamente equivalente. Esto no quiere decir que se prescinde de

los operadores de la lógica temporal sino que se utiliza su equivalente semántico, según la definición de los operadores, gracias a los argumentos temporales.

3.2.2. Arquitectura de los componentes de software utilizados en la implementación

Con base en seleccionar *XSB* por las razones anteriormente explicadas se diseñó la siguiente arquitectura de componentes en capas (en su mayoría de licenciamiento libre excepto *Java*):

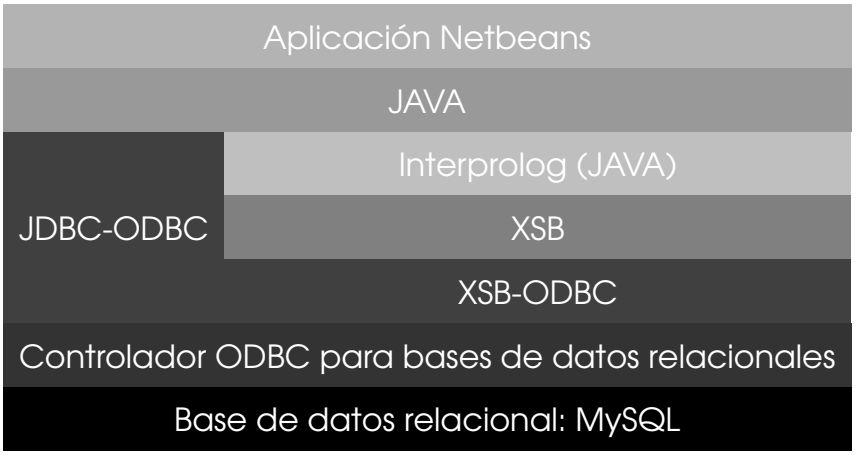

Figura 3.2.1: Arquitectura de componentes del descubridor de patrones

Estos componentes y su integración son explicados a continuación.

3.2.2.1. *Netbeans*, *Java* y *JDBC-ODBC*

Netbeans *NetBeans* se refiere tanto a un marco para el desarrollo de aplicaciones *Java* de escritorio como a un ambiente de desarrollo integrado (IDE, por sus siglas en inglés). Es decir, el IDE *NetBeans* es una aplicación *NetBeans*. Además, el ambiente de desarrollo permite programar con lenguajes como *Java*, *JavaScript*, *PHP*, *Python*, *Ruby*, *Groovy*, *C*, *C++*, *Scala*, *Clojure* y otros.

El IDE *NetBeans* está escrito en *Java* (puesto que las aplicaciones *Netbeans* son aplicaciones *Java*) y puede correr donde sea que esté instalado el intérprete de *Java*. Esto incluye sistemas de explotación tales como *Microsoft Windows*, *Mac OS*, *Linux* y *Solaris*. Se requiere el *Java Development Kit* (JDK) para la funcionalidad de desarrollo en *Java*, aunque no es requerido para el desarrollo en otros lenguajes de programación. *NetBeans* tiene una licencia dual de código abierto. Estas licencias son la *Common Development and Distribution License* (CDDL) y la *GNU Public License 2* con una excepción de *Classpath* (Oracle Corporation, 2011d).

La plataforma *NetBeans* permite el desarrollo de aplicaciones desde un conjunto de componentes modulares de *software* llamados módulos. Las aplicaciones basadas en la plataforma *NetBeans* (incluyendo al IDE *NetBeans*) puede ser extendidas por terceros desarrolladores (Oracle Corporation, 2011c). Este enfoque de módulos fue utilizado en la implementación de *chronos-tmw* para empacar dos bibliotecas de funcionalidad (siendo una *Interprolog*, la cual es explicada más adelante). Además, se incluyen otros módulos de la plataforma *NetBeans*.

Java *Java* es un lenguaje de programación y es el lenguaje en el cual se desarrollan y compilan las aplicaciones *NetBeans* como *chronos-tmw*. Este lenguaje fue originalmente desarrollado por James Gosling de *Sun Microsystems* (que es ahora una subsidaria de *Oracle Corporation*) y fue liberado en 1995 como un componente de la plataforma *Java* de *Sun Microsystems*. El lenguaje deriva su sintáxis de *C* y *C++* pero posee un modelo de objetos más simple y menos facilidades de bajo nivel.

Aunque las especificaciones, compiladores Java, intérpretes y bibliotecas de clases fueron desarrolladas por *Sun Microsystems* desde 1995; en mayo del año 2007, en cumplimiento con las especificaciones del *Java Community Process*, *Sun Microsystems* relicenció la mayoría de sus tecnologías *Java* bajo la licencia *GNU General Public*

License. Otros han desarrollado implementaciones alternativas de estas tecnologías tales como el compilador *GNU* para *Java*, el *GNU Classpath* y *Dalvik*.

JDBC-ODBC Conectividad a Bases de Datos Java (*JDBC*, por sus siglas en inglés) es una interfaz de programación de aplicaciones para el lenguaje de programación *Java* que define como un cliente de bases de datos puede tener acceso a bases de datos. Provee métodos para consultar y actualizar bases de datos; además, está orientado a bases de datos relacionales. Un puente *JDBC* a *ODBC* (explicado a continuación) habilita conexiones hacia cualquier fuente de datos accesible por *ODBC* en el ambiente *JVM* huésped.

En *chronos-tmw* la funcionalidad *JDBC-ODBC* está provista por medio de un módulo *NetBeans*.

3.2.2.2. *ODBC*

Conectividad Abierta a Bases de Datos (*ODBC*, por sus siglas en inglés) es una especificación abierta para proveer a los desarrolladores de aplicaciones, una interfaz predecible con la cual se puede acceder fuentes de datos y aliviar la necesidad de los desarrolladores de aprender múltiples interfaces. *ODBC* es entonces una interfaz universal de acceso a datos. Las fuentes de datos accesibles incluyen servidores de Lenguaje Estructurado de Consultas (SQL, por sus siglas en inglés) o cualquier fuente de datos con un controlador *ODBC*. Con *ODBC*, los desarrolladores de aplicaciones pueden acceder, ver y modificar concurrentemente bases de datos múltiples y diversas.

El uso de *ODBC* en la aplicación *chronos-tmw* significa que debe existir en el sistema de explotación controladores *ODBC* para la base de datos relacional que se desee acceder. Esto es un requisito obligatorio para el uso de la aplicación.

3.2.2.3. *Interprolog*, *XSB Prolog* y *XSB-ODBC*

Interprolog En *chronos-tmw*, *Interprolog* provee integración con programación lógica. El sitio de *InterProlog* (Declarativa - Serviços de Informática, Lda. 2011) describe a esta biblioteca de funcionalidad de código abierto *Java* como una parte frontal y mejora funcional para varios *Prolog* estándar que corren en *Microsoft Windows*, *Linux* and *Mac OS X*. Actualmente apoya el uso de varios motores de programación lógica de código abierto: *XSB Prolog* de los EE.UU. (el más declarativo), *SWI-Prolog* de los Países Bajos (el mejor ambiente) y *YAP Prolog* de Portugal (el más rápido). *InterProlog* incluye ayudas en la visualización de términos *Prolog* y ejemplos de programación, concretamente un editor gráfico de rompecabezas Sudoku y un solucionador de los rompecabezas.

InterProlog le provee a *Java* la habilidad de llamar cualquier meta *Prolog* a través de un objeto *PrologEngine*. También le provee a *Prolog* la habilidad de invocar cualquier método *Java* a través de un predicado *javaMessage* que permite pasar virtualmente cualquier objeto *Java* y términos *Prolog* entre ambos lenguajes con una única instrucción.

Se apoya la recursión mutua y los multihilos *Java*. La reflexión *Java*, los mecanismos de serialización *Java* y las fortalezas naturales de *Prolog* se combinan para proveer flexibilidad y dinamismo. Así, en lugar de parecer una interfaz *Prolog*/*C* que simula una orientación a objetos, *InterProlog* más bien provee una interfaz de programación de aplicaciones que iguala objetos a términos induciendo un estilo de programación más conciso y declarativo.

En *chronos-tmw* se integra *InterProlog* como un módulo *NetBeans* que incluye a la biblioteca Java. Además, este módulo es una dependencia de la aplicación *chronos-tmw*.

XSB Prolog *XSB* es un sistema de programación lógica y base de datos deductiva para *Unix* y *Microsoft Windows*. Está siendo desarrollado en algunas instituciones que incluyen el departamento de ciencias de la computación de *Stony Brook University*, la *Universidade Nova de Lisboa*, a *XSB Inc.*, la *Katholieke Universiteit Leuven* y la *Uppsala Universitet* (XSB Group, 2011b).

Tiene una buena base de características que se encuentran documentadas en dos manuales: "Manual del programador" (Swift et al. 2011) y "Bibliotecas, interfaces y paquetes" (XSB Group, 2011a). Entre esta base de características, de particular importancia en la implementación de *chronos-tmw* es la interfaz *ODBC*, la cual además traduce programación lógica a SQL. Esta interfaz se detalla un poco más adelante.

XSB está disponible en forma de código fuente, así que para plataformas tanto *Microsoft Windows* como *Unix* es necesario compilar el *software*. Durante el desarrollo de *chronos-tmw* se experimentó con las versiones 3.1 a 3.3. *XSB* sigue siendo mantenido y es un proyecto activo.

La licencia de *XSB Prolog* es la *GNU General Public License*.

XSB-ODBC La interfaz *XSB-ODBC* es un subsistema que permite a los usuarios de *XSB Prolog* acceder a bases de datos por medio de conexiones *ODBC*. La interfaz permite acceder datos en cualquier Sistema de Administración de Base de Datos (DBMS, por sus siglas en inglés) que cumpla con *ODBC*. Utilizando esta interfaz uniforme, la información en diferentes DBMS puede ser accesada como si existiera en hechos *Prolog*. La interfaz *XSB-ODBC* provee a los usuarios con tres niveles de interacción: un nivel SQL, un nivel de relación y un nivel de vista. El nivel SQL permite a los usuarios escribir declaraciones SQL explícitas las cuales son pasadas a la interfaz para recuperar datos de una base de datos con la cual hay conexión. El nivel de relación permite a los usuarios declarar predicados *XSB* que conectan

a tablas individuales en una base de datos conectada con la aplicación y los cuales al ser ejecutados apoyan una recuperación de una tupla a la vez, desde la tabla de la base de datos. El nivel de vista permite a los usuarios utilizar una consulta *XSB* compleja, incluyendo conjunciones, negación y sumarizaciones para especificar una consulta de base de datos. (XSB Group, 2011a)

En *chronos-tmw* se hace utilización de esta interfaz a nivel de relación. Por el momento, la aplicación permite buscar patrones secuenciales únicamente sobre una sola tabla.

3.2.2.4. MySQL

MySQL es un DBMS relacional y es el que se utilizó para las pruebas con *chronos-tmw*, aunque como se desprende de la utilización de *ODBC*, el uso de otros DBMS relacionales sería posible. El proyecto de desarrollo *MySQL* ha colocado su código fuente disponible bajo los términos de la licencia *GNU General Public License*, además de una variedad de acuerdos comerciales alternativos (Oracle Corporation, 2011b). *MySQL* era propiedad de la empresa sueca *MySQL AB*, la cual es actualmente propiedad de *Oracle Corporation*.

Es usual que proyectos de código abierto o libre que requieran un DBMS utilicen *MySQL*. En el caso de la experimentación con *chronos-tmw* se utilizó la versión 14.14, distribución 5.5.8, compilada para una arquitectura de 64 *bits*.

MySQL/ODBC Dado que *chronos-tmw* ha sido probado con *MySQL* es necesario contar con un controlador *ODBC* para *MySQL*. Por ello se utilizó *MySQL Connector/ODBC*. Este último es el nombre para la familia de controladores *MySQL ODBC* (previamente llamados *MyODBC drivers*) que proveen acceso a bases de datos *MySQL* utilizando la interfaz de programación de aplicaciones *ODBC*. *MySQL*

Connector/ODBC provee tanto interfaces basadas en controlador y nativa para la base de datos *MySQL* que apoyan enteramente la funcionalidad de *MySQL*, incluyendo procedimientos almacenados, transacciones y, con *Connector/ODBC* 5.1, total cumplimiento con el conjunto de caracteres *Unicode*. (Oracle Corporation, 2011a)

3.2.3. Implementación

Los componentes mencionados en la sección anterior fueron utilizados para crear una aplicación *Netbeans* que carga un módulo a la medida que entrega la funcionalidad de minería de patrones secuenciales. Este módulo programado en *Java* es quién amalgama los diferentes componentes mencionados. Se pueden encontrar más detalles sobre esta implementación en el apéndice B.

Capítulo 4

Casos de estudio y análisis de resultados

De manera general, se dificulta conseguir conjuntos de datos para hacer pruebas de minería de patrones secuenciales. Efectivamente, existen bastantes conjuntos de datos para hacer análisis de reglas de asociación pero sin línea temporal como los datos disponibles en *Frequent Itemset Mining Dataset Repository* (`http://fimi.ua.ac.be/`). También existen conjuntos de datos de series de tiempo como los disponibles en *UCR Time Series Classification/Clustering Page* (`http://www.cs.ucr.edu/~eamonn/time_series_data/`) que son datos temporales, pero no discretos y de una única línea temporal. Esta dificultad ha sido señalada en Omari (2008), Omari et al. (2008). Por otro lado, los datos no sólo deben tener una dimensión temporal sino también una dimensión que permite identificar diferentes secuencias o líneas de tiempo. Así, frente a esta dificultad se decidió por dos casos de prueba:

- El primer caso de prueba consiste en un caso ilustrativo. Este caso ilustrativo es el mismo ejemplo mostrado en Agrawal y Srikant (1994b).

- Como segundo caso de prueba se utilizan datos artificiales generados por una herramienta de generación de datos artificiales.

4.1. Caso ilustrativo

Este caso consiste en un base de datos pequeña. Como se decía, este caso de ejemplo es el mismo disponible en Agrawal y Srikant (1994b). La importancia de este primer caso es mostrar el funcionamiento del algoritmo.

Cuadro 4.1.1: Base de datos trivial

Tiempo	**Elemento (artículo, síntoma, etc.)**	**Identificador de secuencia (cliente, paciente, etc.)**
1	10	2
1	20	2
2	90	5
3	30	2
4	40	2
4	60	2
4	70	2
5	30	4
5	30	3
5	50	3
5	70	3
5	30	1

Tiempo	Elemento (artículo, síntoma, etc.)	Identificador de secuencia (cliente, paciente, etc.)
6	90	1
6	40	4
6	70	4
7	90	4

4.1.1. Paso 1 del algoritmo

Con esta base de datos tendríamos las siguientes secuencias (que se obtienen utilizando un predicado *Prolog*):

```
?- concrete_sequence_database(SequenceId, Sequence).
SequenceId = 1,
Sequence = [ (5, [30]), (6, [90])] ;
SequenceId = 2,
Sequence = [ (1, [10, 20]), (3, [30]), (4, [40, 60, 70])] ;
SequenceId = 3,
Sequence = [ (5, [30, 50, 70])] ;
SequenceId = 4,
Sequence = [ (5, [30]), (6, [40, 70]), (7, [90])] ;
SequenceId = 5,
Sequence = [ (2, [90])].
```

Nótese como a diferencia de la base de datos de secuencias de Agrawal y Srikant (1994b) en esta se incluye un parámetro temporal que indica el tiempo en que ocurren los elementos (o eventos).

La base de datos de secuencias se genera en forma de hechos (cláusulas básicas) en el entorno de programación lógica. Por ello, en el modelo realizado, ésta se llama base de datos concreta de secuencias (*concrete_sequence_database*). Es posible, también, declarar la base de datos de secuencias de manera deductiva. La regla

(cláusula de programa) mostrada a continuación produce de manera deductiva la misma base de datos de secuencias vista. En dicha regla, *Relation* se refiere a un símbolo de relación o tabla (un símbolo de predicado en términos de programación lógica). *Relation*, además, debe ser de la forma *relacion(Tiempo, Elemento, IdentificadorDeSecuencia)*.

```
sequence_database(Relation, SequenceId, Sequence) :-
  setof(
    (Time, Items),
    Time^Item^X^Y^(
      call(Relation, X, Y, SequenceId),
      setof(Item, call(Relation, Time, Item, SequenceId), Items)
    ),
    Sequence
  ).
```

La utilización de una regla para la inferencia de cada secuencia es altamente ineficiente. El número de inferencias que se realizan para obtener cada secuencia con dicha regla crece muy rápido con respecto al tamaño de la base de datos original. Por ello, la conclusión de crear la base de datos de secuencias como hechos es obvia.

Por último, cada secuencia de esta base de datos de secuencias es un modelo tal como se definió en la definición 2.4.7. Cada parámetro temporal puede ser considerado como un número de mundo. Así, para la secuencia 1 se puede decir que $M_1, w_5 \Vdash [30]$, puesto que 30 es verdadero en el tiempo 5 de dicha secuencia, o se puede decir que $M_1, w_5 \Vdash \langle F \rangle [90]$, puesto que [90] es verdadero en el mundo 6 de dicha secuencia. En conclusión, tenemos una estructura que nos permite hacer uso de la lógica temporal.

4.1.2. Paso 2 del algoritmo

En este conjunto de datos buscando un apoyo mínimo del 25 % (o sea, un mínimo de 1,25 secuencias que apoyen el patrón) se tienen los siguientes patrones atómicos (conjuntos de elementos grandes o secuencias tamaño 1): `[[30], [40], [40,70], [70], [90]]`. Efectivamente, es fácil discernir que el conjunto `[30]` está en 4 secuencias, el conjunto `[40]` está en 2 secuencias, el conjunto `[40, 70]` está en 2 secuencias, el conjunto `[70]` está en 3 secuencias y el conjunto `[90]` está en 3 secuencias.

4.1.3. Paso 3 del algoritmo

En este paso se genera el espacio de búsqueda mostrado en la figura 4.1.1. En dicha figura, como es posible notar, algunas fórmulas de lógica temporal están tachadas con una barra oblicua. Esto indica que en dicho nodo se podó el espacio de búsqueda pues el número de secuencias que contienen dicha fórmula no es mayor o igual a 1,25 (el número mínimo de secuencias que deben contener el patrón para un apoyo en los datos de 25 %) y debido al principio APRIORI se sabe que ningún super-patrón contendrá un sub-patrón con un apoyo menor.

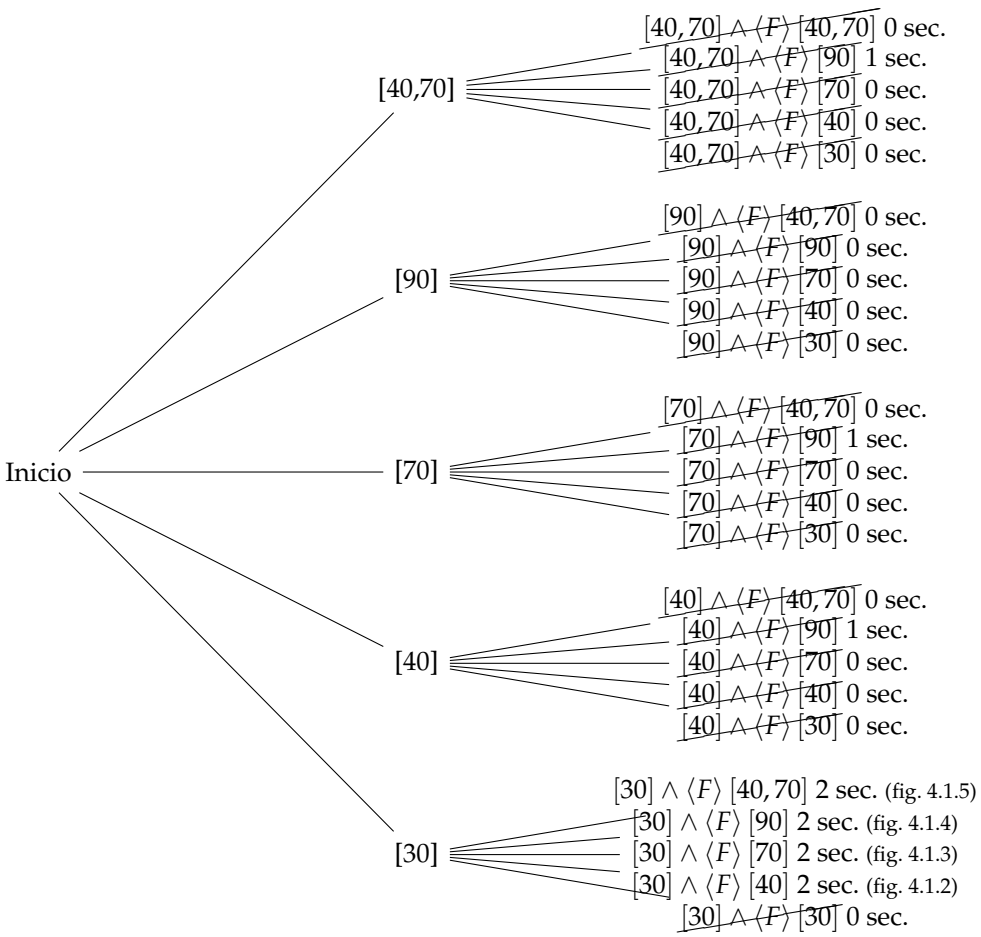

Figura 4.1.1: Espacio de búsqueda del algoritmo para el caso ilustrativo. Apoyo en los datos de 0,25 (al menos 1,25 secuencias de 5).

Es posible notar en la figura 4.1.1 que, con la base de datos de ejemplo, el algoritmo comienza por el conjunto de elementos frecuente $[40, 70]$. Efectivamente, este conjunto de elementos aparece en 2 secuencias (más del número mínimo requerido de 1,25). Dado que la búsqueda se hace en profundidad primero, a partir del conjunto de elementos mencionado, se construye la fórmula temporal $[40, 70] \wedge \langle F \rangle [40, 70]$. Dicha fórmula es apoyada por 0 secuencias así que se desecha. Además, no se explorará más dicha rama del espacio de búsqueda debido al principio APRIORI. Se continúa, entonces, añadiendo otro elemento frecuente a $[40, 70]$, para obtener la fórmula

$[40, 70] \wedge \langle F \rangle\, [90]$, la cual es apoyada únicamente por 1 secuencia así que se desecha. Se continúa, de manera muy similar, con diferentes fórmulas hasta llegar al conjunto de elementos raíz $[30]$, a partir del cual se genera la fórmula $[30] \wedge \langle F \rangle\, [40, 70]$. Dicha fórmula es verdadera en 2 secuencias. Por lo tanto, se profundiza aún más en esa fórmula, pues es posible que existan super-patrones que tengan el apoyo mínimo de los datos en dicha rama del espacio de búsqueda. En la figura 4.1.5 es posible ver la profundización en la fórmula $[30] \wedge \langle F \rangle\, [40, 70]$. Como se puede notar, en dicha rama de la búsqueda, no se encuentran más fórmulas con el apoyo mínimo. Lo mismo sucede para las fórmulas $[30] \wedge \langle F \rangle\, [90]$, $[30] \wedge \langle F \rangle\, [70]$ y $[30] \wedge \langle F \rangle\, [40]$ cuyos sub-espacios de búsqueda se encuentran en las figuras 4.1.4, 4.1.3 y 4.1.2 respectivamente. La fórmula $[30] \wedge \langle F \rangle\, [30]$, en la figura 4.1.1, no es de interés y con ella se termina la búsqueda completa.

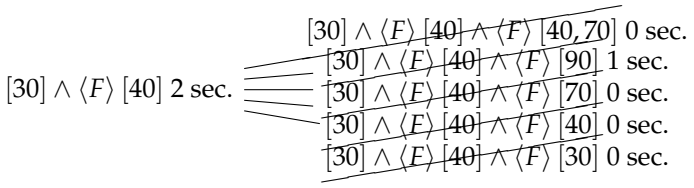

Figura 4.1.2: Continuación del espacio de búsqueda mostrado en la figura 4.1.1.

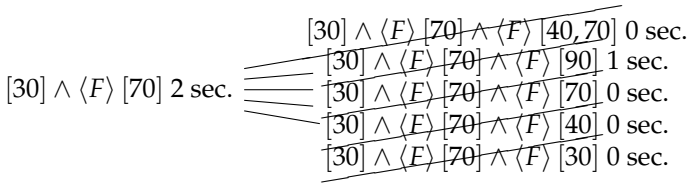

Figura 4.1.3: Continuación del espacio de búsqueda mostrado en la figura 4.1.1.

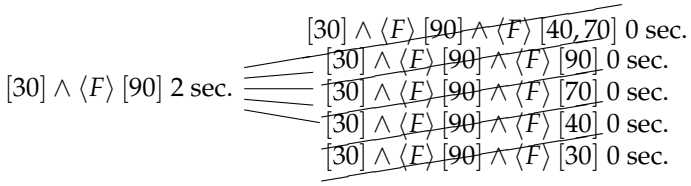

Figura 4.1.4: Continuación del espacio de búsqueda mostrado en la figura 4.1.1.

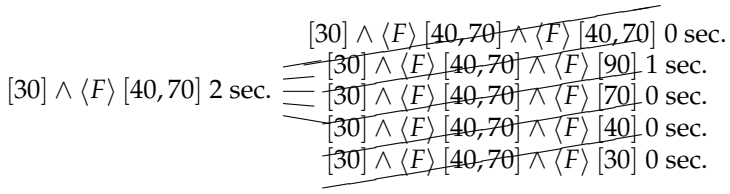

Figura 4.1.5: Continuación del espacio de búsqueda mostrado en la figura 4.1.1.

Así, en base al espacio de búsqueda mostrado en la figura 4.1.1 el algoritmo produce los patrones en el siguiente cuadro[1]:

0.4000: [30] and f [90]
0.4000: [30] and f [70]
0.4000: [30] and f [40,70]
0.4000: [30] and f [40]

Cuadro 4.1.2: Patrones secuenciales con apoyo mayor o igual a 25 %.

4.1.4. Paso 4 del algoritmo

Se eliminan, en este paso, los patrones contenidos en otros patrones con lo cual se obtienen los patrones secuenciales del siguiente cuadro:

[1]La notación utilizada es la notación de salida de *chronos-tmw*. "[30] and f [90]" se lee "30 y alguna vez en el futuro 90". 0.400 implica que el patrón tiene un 40 % de apoyo.

0.4000: [30] and f [90]
0.4000: [30] and f [40,70]

Cuadro 4.1.3: Patrones secuenciales maximales con apoyo mayor o igual a 25 %.

4.2. Caso con datos artificiales

El segundo caso de prueba del modelo y de su implementación en *chronos-tmw* se ejecutó con datos artificiales o sintéticos. El objetivo principal es el de evaluar eficacia.

4.2.1. Generación de datos artificiales

De manera general, se dificulta conseguir conjuntos de datos para hacer pruebas de minería de patrones secuenciales. Esta dificultad de encontrar datos con una dimensión temporal y con un identificador de secuencias (como el identificador de cliente), ha sido señalada en otras investigaciones, entre ellas Omari (2008), Omari et al. (2008). Así, la realización de pruebas con datos artificiales generados específicamente con las características requeridas se asevera útil. Por ello, se consideró válido utilizar programas de terceros como los mencionados en Agrawal y Srikant (1994b), Cooper y Zito (2007), Omari (2008), Omari et al. (2008), Adä y Berthold (2010). Sin embargo, lo que se notó al realizar esta investigación es que las herramientas existentes para generación de datos sintéticos o artificiales suelen ser específicas a tareas diferentes de minería de datos. Para la tarea de búsqueda de patrones secuenciales también son escasas estas herramientas o utilidades. También pudo ser válido utilizar conjuntos de datos reales siempre y cuando cumplieran con las características requeridas para el tipo de minería de datos de la investigación.

Se hizo un esfuerzo grande en generar datos por medio de la herramienta Quest[2]. Sin embargo, la cantidad de esfuerzo hecha no era suficiente por la cantidad de errores en la aplicación. Se decidió desechar la utilización de Quest como parte de la investigación.

Dado que Quest no mostró ser de utilidad para la generación de datos artificiales, se hizo un esfuerzo en desarrollar un generador sencillo de datos artificiales para minería de patrones secuenciales. El desarrollo se realizó en lenguaje *Perl*, sin embargo, el generador sencillo mostró no ser suficiente, pues la generación de datos es un campo de investigación en sí mismo. Así, también se desechó esta alternativa.

Fue posible obtener la herramienta *TARtool* (Omari et al. 2008). Esta herramienta aunque no está disponible en Internet fue facilitada amablemente por su creador luego de contactarle. A pesar de que la herramienta *TARtool* agrega marcas de tiempo a una generación artificial de transacciones de compras de clientes, no fue útil para las pruebas de esta investigación pues no agrega identificadores de secuencias, o en este caso específico, identificadores de clientes.

Inclusive, se evaluó utilizar los datos que genera la herramienta planteada por Cooper y Zito (2007). Sin embargo, se descubrió que la herramienta de Cooper y Zito únicamente genera datos para reglas de asociación sin línea temporal. Es decir, no tiene marcas de tiempo ni identificadores de secuencia, i.e. clientes. De nuevo, se desechó la utilización de esta otra herramienta en la línea de investigación.

Finalmente se decidió utilizar la herramienta *Konstanz Information Miner* (KNIME, por sus siglas en alemán). Esta es una herramienta de minería de datos orienta-

[2]La aplicación QUEST se encontraba disponible inclusive hasta el año 2010 en la dirección `http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html`. Sin embargo, esta página ya no está disponible. Ahora es posible encontrar la aplicación en el archivo de Internet en la dirección `http://web.archive.org/web/http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html` o con modificaciones en las direcciones `https://sourceforge.net/projects/ibmquestdatagen/` y `http://www.cs.loyola.edu/~cgiannel/assoc_gen.html`.

da a flujos de datos modulares. Es posible, una vez instalada la herramienta, instalar módulos de generación de datos. Los detalles se encuentran en Adä y Berthold (2010). Así, con base en esta escogencia se desarrolló la siguiente prueba.

4.2.2. Prueba y resultados obtenidos

El objetivo de esta prueba es medir la eficacia de *chronos-tmw* en descubrimiento de patrones en los datos.

Se diseñó la siguiente metodología para la prueba:

1. Se generaron datos por medio de KNIME. KNIME permite la conexión a un repositorio de flujos de datos prediseñados por la comunidad de utilizadores. El acceso a este repositorio se puede hacer como invitado. En la carpeta *007_ModularDataGeneration* de dicho repositorio se encuentra un flujo de datos llamado *007008_ShoppingBasket* que fue el utilizado para generar datos para las pruebas. Dichos datos pueden ser encontrados en forma resumida y en formato Valores Separados por Coma en *Sourceforge*, sitio de proyectos de *Software* Libre[3]. La primer columna es el identificador de cliente, la segunda columna es el tiempo y la tercer columna es el identificador de producto.

2. Se minaron los datos generados por KNIME utilizando el algoritmo GSP (Agrawal & Srikant, 1995b) disponible en una herramienta de minería de datos de licenciamiento libre llamada *RapidMiner*. Así, se descubrieron los patrones secuenciales en los datos para dos niveles de apoyo: 60 % y 75 %. Los resultados de *RapidMiner* se asumieron correctos.

[3]La localización exacta de dicho archivo es `http://sourceforge.net/projects/chronos-tmw/files/synthetic_data/synthetic_data_generated_using_KNIME_with_007008_ShoppingBasket_out_of_the_box.csv/download`.

3. Se minaron los datos generados por KNIME utilizando *chronos-tmw* en los niveles de apoyo 60 % y 75 %.

4. Se compararon los resultados de *RapidMiner* y de *chronos-tmw* para evaluar la eficacia.

Se utilizan entonces los datos generados por KNIME. Nótese que aquí únicamente se presentan pruebas para un apoyo en los datos de 60 % y 75 % pues para apoyos menores como 45 % el número de patrones encontrados en los datos generados por KNIME es creciente[4]. Aunque esto puede ser interesante en un contexto comercial, en esta investigación puede entorpecer la apreciación del trabajo. Además, para apoyos aún menores, como 30 % y 15 %, el número de patrones y el tiempo de computación crecen aún más.

Para un apoyo de los datos de 60 %, *RapidMiner* encontró un total de 103 patrones maximales. Se debe exceptuar el patrón <item = 45, item = 127> pues este es un patrón tamaño 1 (un conjunto de elementos frecuentes). Así, el número exacto de patrones sería de 102 patrones. Los patrones descubiertos pueden ser encontrados en el apéndice D.

En el siguiente cuadro se pueden apreciar los patrones con un apoyo de los datos mayor o igual a 75 %.

Patrones con apoyo mayor o igual a 75 %
0.750: <item = 152> <item = 92>
0.770: <item = 152> <item = 152>

Cuadro 4.2.2: Patrones secuenciales descubiertos por *RapidMiner* en los datos generados por KNIME.

[4]Para 45 % de apoyo de los datos, *RapidMiner* encuentra un total de 3061 patrones. Eliminando los patrones contenidos en otros se tienen finalmente 1737 patrones.

Para la interpretación de estos dos patrones es necesario saber que en los datos generados por KNIME el elemento 92 se refiere a *pavo* (el ave consumida como carne blanca) y el elemento 152 se refiere a *bio coke* (una bebida carbonatada). Así, estos dos patrones se pueden interpretar como "el cliente compra *bio coke* y alguna vez en el futuro compra pavo" y como "el cliente compra *bio coke* y alguna vez en el futuro vuelve a comprar *bio coke*" respectivamente. El primer patrón podría ser interpretado como una relación entre ambos productos y el segundo patrón puede ser interpretado como una satisfacción con el producto. Ambos patrones pueden ser utilizados en un proceso de toma de decisiones como, por ejemplo, la producción de campañas publicitarias.

Nótese que *RapidMiner* no da como resultado sólo patrones maximales (o sea, patrones que no están contenidos en otro patrón) sino que entrega como resultado todos los patrones. Esto puede ser rápidamente abrumador si los patrones serán leídos por un ser humano en lugar de ser utilizados por algún sistema automático, como por ejemplo, un sitio de comercio electrónico. Así, los patrones con 75 % de apoyo de los datos fueron limpiados a mano y los patrones de 60 % fueron limpiados de sub-patrones por medio del programa especificado en el apéndice C.

Otra característica en *RapidMiner* es el hecho de que entrega secuencias de tamaño 1. Estas son equivalentes a conjuntos de elementos grandes y normalmente no se toman en cuenta en los resultados de búsqueda de patrones secuenciales.

4.2.2.1. Resultados esperados

Se espera, entonces, tras correr *chronos-tmw* encontrar los mismos patrones que encontró *RapidMiner* y comprobar así su eficacia. Se hace únicamente una corrida sobre un único conjunto de datos artificiales, pues estos son 48705 registros que

fueron generados de manera aleatoria. Debido a esta aleatoriedad es improbable que si *chronos-tmw* es incorrecto produzca el mismo resultado que *RapidMiner*.

4.2.2.2. Resultados obtenidos

Los patrones encontrados por *chronos-tmw* coinciden con los patrones encontrados por *RapidMiner*. La salida de *chronos-tmw* con dichos patrones, para el apoyo de 60 %, puede ser encontrada en el apéndice E. En el siguiente cuadro (4.2.3) se puede ver la salida de *chronos-tmw* para el apoyo de 75 %.

```
+-----------------------------------------------------+
| Sequential Patterns Mining with Temporal Logic |
+-----------------------------------------------------+

Support: 0.7500
Creating sequence database, please wait...
Time creating sequence database: %5.818 CPU in 5.988 seconds (97% CPU)

Sequence database: 200 sequences
Creating large-itemsets (1-itemsets), please wait...
Time creating large-itemsets: %32.901 CPU in 33.943 seconds (96% CPU)

Large Itemsets with more than 150.0000 sequences support: 17
Finding patterns, please wait...
Time finding patterns: %9.344 CPU in 9.747 seconds (95% CPU)

Patterns:
0.7700: [152] and f [152]
0.7500: [152] and f [92]
Total patterns: 2
```

Cuadro 4.2.3: Patrones encontrados por *chronos-tmw* en los datos generados por KNIME y con un 75 % de apoyo en los datos.

Este resultado comprueba lo correcto de la programación del algoritmo en el código de *chronos-tmw* y muestra el comportamiento correcto del algoritmo utilizando lógica temporal. La importancia de este resultado es que se mostró eficacia con una cantidad suficiente de registros, en los cuales el ser humano no puede dilucidar nada (ni con la ayuda de una hoja de cálculo).

Capítulo 5

Conclusiones e investigación futura

5.1. Conclusiones

Este trabajo de investigación es un paso más de apropiación del conocimiento y de apropiación tecnológica en el contexto costarricense. Se cumplió con el objetivo de estudiar la lógica modal temporal y con ello se realizó un trabajo de apropiación de conocimiento. Se hizo un trabajo en el área de programación lógica con una arquitectura de componentes de licenciamiento libre, con lo cual se dio indirectamente un proceso de apropiación tecnológica.

En cuanto al estudio de lógica modal se descubrió un marco formal sólido por medio del cual es posible describir y explorar estructuras relacionales. Este marco permite también hacer inferencias sobre estas estructuras. De manera más particular, la lógica modal temporal con una relación de orden, se muestra particularmente útil en dominios que incluyen el tiempo como dimensión. Así, esta se aseveró aplicable en la búsqueda de patrones secuenciales en bases de datos temporales. De utilidad importante fue la relación '<' que permite el acceso entre mundos lógicos. La lógica temporal mostró ser un marco teórico sólido sobre el cual generalizar el descubrimiento de patrones secuenciales.

Se diseñó e implementó un modelo para el descubrimiento de patrones secuenciales en bases de datos utilizando lógica temporal. Este modelo es una modificación del algoritmo original de descubrimiento de patrones secuenciales (Agrawal & Srikant, 1994b). Se reemplazó el paso de descubrimiento de patrones secuenciales con un conteo de fórmulas de lógica temporal que son equivalentes a patrones secuenciales. Este nuevo algoritmo se implementó por medio de lógica temporal. Además, se transformó la base de datos relacional subyacente a una base de datos temporal deductiva por medio de parámetros temporales y la utilización de un motor de programación lógica. La programación lógica, sin embargo, a pesar de ser muy legible por su naturaleza declarativa, no siempre es eficiente. Así, muchas veces fue necesario re-programar las reglas implementadas en *chronos-tmw* para eliminar "cuellos de botella" en términos de rendimiento. Finalmente, ciertas reglas terminaron siendo más complejas y en cierta forma menos declarativas o más "imperativas".

Para las pruebas, se hizo un esfuerzo grande en generar datos por medio de la herramienta Quest. Quest permite generar datos artificiales para tres tareas diferentes de minería de datos: conjuntos frecuentes de elementos grandes sin taxonomías, conjuntos frecuentes de elementos grandes con taxonomías y patrones secuenciales. El código fuente de esta aplicación, tal como se menciona en la sección 2.1 de esta investigación no compila en compiladores modernos. Luego de corregir esta situación con las modificaciones planteadas en el apéndice F y de lograr compilar el código es evidente que existen fallas importantes en lo correcto de la programación de la herramienta y en la generación de datos artificiales para la tarea de descubrimiento de patrones secuenciales (no se evaluó la generación para las otras dos tareas de minería). Uno de los errores más importantes es la referencia a direcciones nulas en diferentes lugares del código fuente. Se hizo un esfuerzo por corregir

la aplicación. Dado que la cantidad de esfuerzo hecha no era suficiente se decidió desechar la utilización de Quest como parte de la investigación.

También se hizo un esfuerzo en desarrollar un generador sencillo de datos artificiales para minería de patrones secuenciales. El desarrollo se realizó en lenguaje *Perl* y arrojó varias conclusiones:

- Tal como se había intuido en base al material bibliográfico (Argüello Venegas, 1997; Cooper & Zito, 2007; Omari et al. 2008; Adä & Berthold, 2010) el tópico de la generación de datos secuenciales es un tema de investigación en sí mismo.
- Producir datos aleatorios no es tarea fácil, pues aún en datos creados a partir de funciones generadoras de números pseudo-aleatorios pueden haber patrones. Inclusive la media y la varianza designada pueden no coincidir con los resultados dependiendo de la cantidad de datos que se produzcan. Las funciones generadoras de números realmente aleatorios tienen un alto costo computacional. Para la producción de datos masivos esto no es práctico.
- Crear los patrones secuenciales a insertar no es difícil, lo difícil es mezclarlos con los datos aleatorios por diversas razones. El patrón puede tener diferente tamaño que la secuencia en la cual se inserta, por lo cual, se deben alinear ambas secuencias de conjuntos de elementos. Cómo se debe realizar esta alineación no es obvio, depende mucho de si ya se insertaron patrones o de los datos aleatorios. Si la secuencia donde se está insertando el patrón no es suficientemente aleatoria se debe aleatorizar, sin embargo, si ya se insertó un patrón anterior se debe tener cuidado con él.

Finalmente, se ejecutó el modelo de descubrimiento de patrones secuenciales sobre un conjunto de datos artificiales almacenados en una base de datos relacional y

generados por la herramienta KNIME (Adä & Berthold, 2010). El análisis de los resultados muestra que la aplicación realizada bajo el modelo diseñado es eficaz en encontrar los patrones secuenciales en los datos. Esto se hizo comparando los resultados de *chronos-tmw* con los resultados de *RapidMiner*. En fin, se mostró que es posible obtener los patrones secuenciales con el modelo diseñado y desarrollado.

Además, a pesar de que la herramienta *chronos-tmw* fue creada bajo una preocupación constante por la eficacia en la tarea de minería, se notó que los patrones conseguidos por la herramienta fueron obtenidos en un tiempo adecuado. Así, aunque existen herramientas de minería de datos más eficientes en términos de tiempo, para la misma tarea de minería de datos, se exploró una área teórica poco revisada a nivel mundial e ignorada en el contexto costarricense.

Fue posible notar, también, que en el ámbito de aplicaciones libres para el descubrimiento de patrones secuenciales aún se puede mejorar. Por ejemplo, *RapidMiner* tiene varias características que lo alejan de una implementación completa de lo mencionado en Agrawal y Srikant (1994b, 1995b). *RapidMiner* entrega en los resultados patrones de tamaño 1 (las secuencias tienen un tamaño mínimo de 2) y tampoco entrega en sus resultados únicamente patrones maximales. Esto podría ser visto como un problema de calidad, es decir, incumplimiento de requerimientos funcionales.

5.2. Investigación futura

Como trabajo futuro de investigación se plantean mejoras de rendimiento u optimizaciones. Se puede mejorar la representación de la base de datos de secuencias, pues tal como se mencionó en la sección 3.1.3.1, no se realizó la asociación de conjuntos de elementos a identificadores. Esta es en términos de rendimiento

una característica importante ya que permite la búsqueda de la existencia de un conjunto de elementos en tiempo constante. Se deja entonces esta característica para mejoras de rendimiento posteriores en el trabajo futuro. Sería importante también explorar la posibilidad de paralelizar la tarea de búsqueda, además de revisar otros algoritmos de búsqueda de conjuntos de elementos frecuentes para mejorar el paso 2 del algoritmo, tal como se menciona en la sección 3.1.3.2. También podría ser útil profundizar en aspectos teóricos como lógica probabilística y funciones de punto fijo. Por último, otro aspecto a considerar es el descubrimiento incremental. Muchos sistemas aprenden patrones de un conjunto relativamente pequeño de los datos (una muestra de tal vez un 1 %), luego prueban ese patrón en los subsiguientes datos, agregando excepciones a la muestra original. Con esta nueva muestra se re-acondicionan los patrones o re-inducen los patrones. De tal manera que no es necesario en general procesar millones de casos.

Un caso de estudio interesante, innovador e importante en un contexto costarricense e inclusive internacional sería el de aplicar el descubrimiento de patrones secuenciales en datos reales, en particular del sector de la salud (expedientes médicos). Por ejemplo, se podría hacer una minería de patrones secuenciales tomando los síntomas y los diagnósticos como los elementos frecuentes, la fecha de la cita como la marca de tiempo y el paciente como el identificador de secuencia. Luego, sería posible realizar procesos de toma de decisiones, por ejemplo, en el área de salud preventiva, interpretando la relación que existe entre síntomas y diagnósticos con otros síntomas y diagnósticos en el futuro. Sin embargo, encontrar datos de calidad de este tipo de aplicación, en un ámbito local o internacional, puede ser motivo de frustración (Ramakrishnan, Hanauer & Keller, 2010). En Junio 2010, se entrevistó al Dr. Víctor Pérez, director médico del Hospital La Católica. Desde su punto de vista y su experiencia en dicha institución médica, en las clínicas privadas

los clientes no vuelven, hay muchas cirugías estéticas (por ejemplo, aumento mamario), los consultorios son alquilados y sólo hay seguimiento del paciente si este "pasa a piso", es decir, si se prepara para cirugía dentro de las instalaciones hospitalarias. Como se puede notar no es posible obtener la historia completa del paciente. Además, las cirugías plásticas no obedecen a razones de salud sino a decisiones personales y aunque la minería puede ser útil en este caso para fines comerciales, no lo es en el ámbito de la salud. En las clínicas y hospitales del Estado pareciera ser que existe una especie de expediente electrónico implementado o en desarrollo. Sin embargo, no fue posible conseguir información. Además, pareciera ser que en algunos hospitales, como el hospital San Juan de Dios, sucede el contrario de las clínicas privadas, se pierde el seguimiento del paciente cuando el paciente "pasa a piso", aunque esta información no es definitiva. Por último, una posibilidad sería la utilización del expediente electrónico del Hospital Clínica Bíblica que pareciera ser más completo y moderno.

Otra línea de investigación interesante, relacionada a la anterior, sería evaluar la calidad de la aplicabilidad de los patrones obtenidos en algún contexto local por medio de una metodología formal cualitativa. La metodología cualitativa se basaría en el criterio de opinión de expertos que hayan sido expuestos a los resultados del descubridor de patrones secuenciales. Se buscaría evaluar lo que los patrones permiten en cuanto a toma de decisiones:

- Entender cuándo y porqué es necesario tomar la decisión.
- Caracterizar la decisión (¿cuál es?, ¿cómo trabajarla?, ¿a quién involucra?).
- Trabajarla (alternativas, entender las posibilidades y probabilidades, escoger una opción).
- Comprometer recursos.

Una interfaz gráfica amigable y agradable es recomendable para la presentación de los resultados de la minería de datos. Esto hace el descubrimiento de patrones más entendible al usuario. También se puede agregar un análisis interactivo de los resultados. Así, una presentación de los patrones descubiertos con alta usabilidad es algo que queda por hacer para continuar este trabajo de investigación.

Por último y en relación a las tecnologías utilizadas para la experimentación de esta tesis como son el marco de aplicación *Netbeans*, *Java*, *Interprolog* y *XSB*, surge una idea de investigación y desarrollo para el futuro que es la creación de herramientas gráfica de extracción, transformación y carga de datos utilizando las tecnologías mencionadas.

Bibliografía

Adä, I. & Berthold, M. R. (2010). The new iris data: modular data generators. En *Proceedings of the 16th acm sigkdd international conference on knowledge discovery and data mining* (pp. 413-422). KDD '10. Washington, DC, USA: ACM.

Agrawal, R., Imielinski, T. & Swami, A. N. (1993). Mining association rules between sets of items in large databases. En P. Buneman & S. Jajodia (Eds.), *Proceedings of the 1993 acm sigmod international conference on management of data, washington, d.c., may 26-28, 1993* (pp. 207-216). ACM Press.

Agrawal, R. & Srikant, R. (1994a). Fast algorithms for mining association rules in large databases. En J. B. Bocca, M. Jarke & C. Zaniolo (Eds.), *Vldb'94, proceedings of 20th international conference on very large data bases, september 12-15, 1994, santiago de chile, chile* (pp. 487-499). Morgan Kaufmann.

Agrawal, R. & Srikant, R. (1994b, octubre). *Mining sequential patterns* (inf. téc. N.º IBM Research Report RJ 9910). San José, California.

Agrawal, R. & Srikant, R. (1995a, marzo). Mining sequential patterns. En *Icde '95: proceedings of the eleventh international conference on data engineering* (pp. 3-14). Washington, DC, USA: IEEE Computer Society.

Agrawal, R. & Srikant, R. (1995b, diciembre). *Mining sequential patterns: generalizations and performance improvements* (inf. téc. N.º IBM Research Report RJ 9994). San José, California.

Argüello Venegas, J. R. (1997). Generación de grandes bases de datos artificiales para evaluación de herramientas de minería de datos. *Ingeniería*, *6*(2), 9-18.

Bertossi, L. (1996, marzo). *Lógica para ciencia de la computación* (Primera Edición) (V. A. Ediciones Universidad Católica de Chile, Ed.). CIP - Pontificia Universidad Católica de Chile.

Chomicki, J. (1994). Temporal query languages: a survey. En *Proceedings of the first international conference on temporal logic* (pp. 506-534). ICTL '94. London, UK: Springer-Verlag.

Cooper, C. & Zito, M. (2007). Realistic synthetic data for testing association rule mining algorithms for market basket databases. En *Pkdd 2007: proceedings of the 11th european conference on principles and practice of knowledge discovery in databases* (Vol. 4702, pp. 398-405). Warsaw, Poland: Springer-Verlag.

Das, S. K. (1992). *Deductive databases and logic programming*. Addison-Wesley.

De Amo, S., Furtado, D. A., Giacometti, A. & Laurent, D. (2004). An apriori-based approach for first-order temporal pattern mining. En *Sbbd* (pp. 48-62).

De Amo, S., Junior, W. P., Giacometti, A. & Clemente, T. G. (2007). Mining temporal relational patterns over databases with hybrid time domains. En *Sbbd* (pp. 347-361).

InterProlog - a Prolog-Java interface. (2011). Recuperado desde `http://www.declarativa.com/interprolog/`

Dehaspe, L. & Toivonen, H. (1999). Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, *3*(1), 7-36.

Dunham, M. H. (2002). *Data mining: introductory and advanced topics*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Fernández, J. L., Manjarrés Riesco, Á. & Diéz Vegas, F. J. (2007). *Lógica computacional* (Departamento de Inteligencia Artificial, Ed.). Escuela Técnica Superior de Ingeniería Informática, UNED. Recuperado desde `http://ogai.name/uned/logica/libro-logica-07.pdf`

Fürnkranz, J. (1997). *Knowledge discovery in chess databases: a research proposal* (inf. téc. N.º OEFAI-TR-97-33).

Galton, A. (2008). Temporal logic. En E. N. Zalta (Ed.), *The stanford encyclopedia of philosophy* (Otoño 2008).

Gregory, M. B. & Shneiderman, B. (2010, mayo). *Shape identification in temporal data sets* (inf. téc. N.º HCIL-2010-05).

Han, J., Pei, J. & Yin, Y. (2000). Mining frequent patterns without candidate generation. En W. Chen, J. F. Naughton & P. A. Bernstein (Eds.), *Proceedings of the 2000 acm sigmod international conference on management of data, may 16-18, 2000, dallas, texas, usa* (pp. 1-12). ACM.

Han, J. & Kamber, M. (2001). *Data mining: concepts and techniques* (A. Bress, Ed.). Morgan Kaufmann Publishers.

Hätönen, K., Klemettinen, M., Mannila, H., Ronkainen, P. & Toivonen, H. (1996). Knowledge discovery from telecommunication network alarm databases. En *Icde '96: proceedings of the twelfth international conference on data engineering* (pp. 115-122). Washington, DC, USA: IEEE Computer Society.

Houtsma, M. & Swami, A. (1993, septiembre). *Set-oriented mining of association rules* (inf. téc. N.º IBM Research Report RJ 9567). San José, California.

Kam, (2000). Discovering temporal patterns for interval-based events.

Letia, I. A., Craciun, F., Köpe, Z. & Lelutiu, A. (2000). First experiments for mining sequential patterns on distributed sites with multi-agents. En *Ideal '00: proceedings of the second international conference on intelligent data engineering*

and automated learning, data mining, financial engineering, and intelligent agents (pp. 187-192). London, UK: Springer-Verlag.

Liu, D., Xiong, X., DasGupta, B. & Zhang, H. (2006). Motif discoveries in unaligned molecular sequences using self-organizing neural network. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 17-919.

Mannila, H., Toivonen, H. & Verkamo, A. I. (1995). Discovering frequent episodes in sequences. En *In proceedings of the 1st international conference on knowledge discovery in databases and data mining* (pp. 210-215).

Mannila, H., Toivonen, H. & Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data Mining Knowledge Discovery*, *1*(3), 259-289. doi:10.1023/A:1009748302351

Mannila, H., Toivonen, H. & Verkamo, A. I. (1994). Efficient algorithms for discovering association rules. En *Kdd workshop* (pp. 181-192).

Masseglia, F., Cathala, F. & Poncelet, P. (1998). The psp approach for mining sequential patterns. En *Pkdd '98: proceedings of the second european symposium on principles of data mining and knowledge discovery* (pp. 176-184). London, UK: Springer-Verlag.

Mooney, C. H. & Roddick, J. F. (2004). Mining relationships between interacting episodes. En *Proceedings of the 4th siam international conference on data mining (sdm'04). siam* (pp. 21-8).

Omari, A. (2008). *Data mining for retail website design and enhanced marketing*. (Tesis doctoral, Mathematisch-Naturwissenschaftlichen Fakultät, Heinrich-Heine-Universität Düsseldorf, Alemania).

Omari, A., Langer, R. & Conrad, S. (2008). Tartool: a temporal dataset generator for market basket analysis. En *Proceedings of the 4th international conference on advanced data mining and applications* (pp. 400-410). ADMA '08. Chengdu, China: Springer-Verlag.

MySQL :: MySQL 5.1 Reference Manual :: 20.1 MySQL Connector/ODBC. (2011a). Recuperado desde http://dev.mysql.com/doc/refman/5.1/en/connector-odbc.html

MySQL :: MySQL 5.5 Reference Manual :: 1.3.1 What is MySQL? (2011b). Recuperado desde http://dev.mysql.com/doc/refman/5.5/en/what-is-mysql.html

NetBeans Platform Learning Trail. (2011c). Recuperado desde http://netbeans.org/kb/trails/platform.html

The Common Development and Distribution License (CDDL) and GNU Public License v.2 w/Classpath Exception. (2011d). Recuperado desde http://netbeans.org/about/legal/license.html

Padmanabhan, B. & Tuzhilin, A. (1996). Pattern discovery in temporal databases: a temporal logic approach. En *Kdd* (pp. 351-354).

Park, J. S., Chen, M.-S. & Yu, P. S. (1995). An effective hash based algorithm for mining association rules. *Sigmod Record*, *24*, 175-186.

Rainsford, C. P. & Roddick, J. F. (1999). Database issues in knowledge discovery and data mining. *Journal of Information Systems*, *6*, 101-128.

Ramakrishnan, N. & Grama, A. Y. (1999). Data mining-guest editors' introduction: from serendipity to science. *Computer*, *32*(8), 34-37. doi:10.1109/2.781632

Ramakrishnan, N., Hanauer, D. & Keller, B. (2010). Mining electronic health records. *Computer*, *43*, 77-81. doi:10.1109/MC.2010.292

Ramirez, J. C. G., Cook, D. J., Peterson, L. L. & Peterson, D. M. (2000a, diciembre). An event set approach to sequence discovery in medical data. *Intell. Data Anal. 4*, 513-530.

Ramirez, J. C. G., Cook, D. J., Peterson, L. L. & Peterson, D. M. (2000b, julio). Temporal pattern discovery in course-of-disease data. *Engineering in Medicine and Biology Magazine, IEEE*, *19*(4), 63-71. doi:10.1109/51.853483

Roddick, J. F., Spiliopoulou, M. & Society, I. C. (2002). A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering*, *14*, 750-767.

Sagonas, K., Swift, T. & Warren, D. S. (1994). Xsb as an efficient deductive database engine. En *Proceedings of the 1994 acm sigmod international conference on management of data* (pp. 442-453). SIGMOD '94. Minneapolis, Minnesota, United States: ACM.

Srikant, R. (1995). *Quest synthetic data generation code*. Recuperado desde `http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html`

Srikant, R. & Agrawal, R. (1996, marzo). Mining sequential patterns: generalizations and performance improvements. En *Edbt '96: proceedings of the 5th international conference on extending database technology* (pp. 3-17). Avignon, France: Springer-Verlag.

Swift, T., Warren, D. S., Sagonas, K., Freire, J., Rao, P., Cui, B., ... Kifer, M. (2011, abril). *The xsb system, version 3.3, volume 1 - programmer's manual*. Recuperado desde `http://xsb.sourceforge.net/manual1/manual1.pdf`

SWI-Prolog's home. (2011). Recuperado desde `http://www.swi-prolog.org/`

TASDA. (2002, agosto). Tasda discussion - why temporal kdd? Recuperado desde `http://tasda.elibel.tm.fr/discussion/why.mhtml`

Torres, L. C. (2007). Anexo c. lógica temporal, En *Inteligencia artificial. conceptos básicos* (pp. 273-287). Universidad Nacional de Colombia. Recuperado desde `http://dis.unal.edu.co/profesores/lucas/iartificial/IA0026l.pdf`

Tseng, V. S. & Lin, K. W. (2007, octubre). Energy efficient strategies for object tracking in sensor networks: a data mining approach. *J. Syst. Softw. 80*, 1678-1698. doi:`10.1016/j.jss.2006.12.561`

Venema, Y. (2001). Temporal logic. En L. Goble (Ed.), *The blackwell guide to philosophical logic* (Cap. 10, pp. 203-223). Malden, USA: Blackwell Publishers.

Vilalta, R. & Ma, S. (2002). Predicting rare events in temporal domains. En *Icdm '02: proceedings of the 2002 ieee international conference on data mining (icdm'02)* (p. 474). Washington, DC, USA: IEEE Computer Society.

Wang, W., Yang, J. & Yu, P. S. (2001). Meta-patterns: revealing hidden periodic patterns. En *Ibm research report* (pp. 550-557).

Winston, P. (1984). *Artificial intelligence* (Addison-Wesley, Ed.).

The XSB System, Version 3.3, Volume 2 - Libraries, interfaces and packages. (2011a, abril). Recuperado desde `http://xsb.sourceforge.net/manual2/manual2.pdf`

XSB. (2011b). Recuperado desde `http://xsb.sourceforge.net/`

Yang, J., Wang, W. & Yu, P. S. (2003). Stamp: on discovery of statistically important pattern repeats in long sequential data. En *Proceedings of the 3rd siam international conference on data mining (sdm'03* (pp. 224-238). SIAM.

Zhao, Q. & Bhowmick, S. S. (2003). Sequential pattern matching: a survey.

Apéndice A

Más sobre lógica modal

A.1. Otras definiciones y teoremas de la lógica modal

Validez En lógica de proposiciones, la satisfacción de una fórmula se produce, o no, en una línea de la tabla de verdad. La fórmula es válida (una tautología) si se satisface en todas las líneas. Análogamente, una fórmula modal es válida si se satisface en toda interpretación posible.

Definición A.1.1. (Validez) Una fórmula es válida si se satisface en todo mundo de todo modelo (Fernández et al. 2007).[1]

Una fórmula válida se satisface en todo caso: en todo dominio W y con cualquier relación R (o sea, en todo marco), cualquiera que sea la asignación v y el mundo w donde se evalúa. Así, la validez no puede depender de una elección particular de estas opciones: depende sólo de la propia estructura de la fórmula y de la semántica de las fórmulas modales.

Teorema A.1.2. *Toda tautología proposicional clásica es una fórmula válida (Fernández et al. 2007).*

[1] **Notación:** para expresar la validez de una fórmula ϕ se utiliza la notación $\models \phi$.

Considérese un marco y un mundo cualesquiera. Sin importar cuál sea la asignación (es decir, los valores de las letras proposicionales en ese mundo), la fórmula (por ser tautología) es verdadera en él.

Las tautologías no son las únicas fórmulas válidas. La fórmula del siguiente teorema es un ejemplo de ello. Esa fórmula es suficientemente importante como para recibir un nombre propio. Así, se le denomina K en honor del lógico Saúl Kripke.

Teorema A.1.3. *La fórmula* $K := (\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q))$ *es una formula válida (Fernández et al. 2007).*

Demostración. Para demostrar $\models K$ se precisa demostrar $M, w \Vdash \phi$ para todo mundo w de todo modelo M. El esquema de demostración siguiente se refiere a un mundo w y a un modelo M cualesquiera. Como no se utilizan características particulares de los mismos, resulta un esquema de demostración correcto para todo mundo y todo modelo.

La fórmula K es un condicional. El único caso en que un condicional no se satisface es cuando sí lo hace el antecedente pero no el consecuente. Pero esto no se produce para K. Si se supone la satisfacción del antecedente esto nos conduce lógicamente a la satisfacción del consecuente.

Así, supóngase $M, w \Vdash \Box(p \rightarrow q)$. Entonces (1): se satisface $(p \rightarrow q)$ para todo mundo de M accesible desde w. Por otro lado, el consecuente de K es asimismo un condicional. Sólo es falso si se satisface su antecedente pero no su consecuente. Supóngase también, entonces, $M, w \Vdash \Box p$. Es decir, (2): se satisface p para todo mundo M accesible desde w.

De (1) y (2) se sigue que se satisface q en todo mundo accesible desde w, y por tanto $M, w \Vdash \Box q$ (Fernández et al. 2007). □

Algunas fórmulas, sin llegar a ser válidas, pueden aún cumplir propiedades menos exigentes. Por ejemplo, pueden ser verdaderas en todo mundo de cierto modelo M, o de cierta familia de modelos. En estos casos, $W, R, v, w \Vdash \phi$ aún se requiere para todo w pero ya sólo para ciertos modelos, ciertas elecciones de marco y asignación. Un caso particular se produce cuando esa familia de modelos coincide con todos los modelos basados en cierto marco o en un conjunto de ellos.

Definición A.1.4. (Validez en un conjunto de modelos) (Fernández et al. 2007)[2]

- Una fórmula ϕ es válida en un modelo M si es verdadera en todo mundo de M.
- Una fórmula ϕ es válida en un conjunto de modelos $\mathcal{M}$ si es válida en todo modelo $M \in \mathcal{M}$.

Como caso particular, resulta de especial interés la validez sobre todos los modelos de un marco o de un conjunto de marcos:

- Una fórmula ϕ es válida en un marco F si es válida en todo modelo basado en F.
- Una fórmula ϕ es válida en un conjunto de marcos $\mathcal{F}$ si es válida en todo marco $F \in \mathcal{F}$.

Teorema A.1.5. *Suponga que ψ es una instancia por sustitución uniforme de ϕ. Para cualquier conjunto de marcos $\mathcal{F}$, si $\mathcal{F} \Vdash \phi$ entonces $\mathcal{F} \Vdash \psi$ (Fernández et al. 2007).*

Nótese que el conjunto de 'todos los modelos posibles' se puede definir como el conjunto de 'los modelos basados en todos los marcos posibles'. Las tautologías y

[2] **Notación:** las fórmulas válidas en un modelo, un conjunto de modelos, un marco y un conjunto de marcos se denotan, respectivamente, como $M \Vdash \phi$, $\mathcal{M} \Vdash \phi$, $F \Vdash \phi$ y $\mathcal{F} \Vdash \phi$.

K son válidas sobre este $\mathcal{F}$ global. Por lo tanto, todas sus instancias por sustitución seguirán siendo válidas sobre este conjunto $\mathcal{F}$. Es decir, las instancias por sustitución de tautologías y de la fórmula K resultan ser válidas, con lo cual, son válidas en cualquier conjunto de modelos $\mathcal{M}$.

Teorema A.1.6. *Si $\mathcal{M}$ es una colección de modelos, el conjunto de fórmulas válidas en $\mathcal{M}$ (Fernández et al. 2007):*

1. incluye todas las instancias de tautologías
2. incluye todas las formulas de la forma $(\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi))$
3. si incluye ϕ y $(\phi \rightarrow \psi)$, entonces incluye ψ
4. si incluye ϕ, entonces incluye $\Box\phi$

Además, si $\mathcal{F}$ es una colección de marcos, el conjunto de fórmulas válidas sobre $\mathcal{F}$ es cerrado respecto a la sustitución.

Las dos primeras categorías son fórmulas válidas, en general, y por tanto fórmulas válidas en cualquier conjunto restringido de modelos. El conjunto $\mathcal{M}$ puede contar además con otras fórmulas válidas en él (aunque no válidas en general). Lo que garantizan las dos últimas condiciones es que en $\mathcal{M}$ el conjunto total de sus fórmulas válidas (las generales y las particulares) es cerrado respecto a ellas. En otras palabras:

1. en cada mundo en que ϕ y $(\phi \rightarrow \psi)$ se satisfacen también se satisface ψ. Si se supone que ϕ y $(\phi \rightarrow \psi)$ se satisfacen en todo mundo de todo modelo de $\mathcal{M}$, igualmente ocurre con ψ.

2. si ϕ fuese una fórmula cualquiera, puede satisfacerse ϕ sin que se satisfaga $\Box\phi$. Ahora bien, si se parte de que ϕ es válida (verdadera en todo mundo), lo es en todos los vecinos accesibles, luego se satisface $\Box\phi$ en todo mundo (para el mismo conjunto de modelos en que ϕ sea válida).

Además, sólo en el caso de que se considere un conjunto de modelos basados en marcos, la sustitución uniforme preserva la validez sobre este conjunto.

Dado un conjunto de marcos $\mathcal{F}$, se denomina la lógica basada en $\mathcal{F}$ al conjunto de fórmulas válidas en $\mathcal{F}$.

Consecuencia Se dice que un conjunto Γ de fórmulas se satisface en un mundo w del modelo M si $M, w \Vdash \gamma$ y para toda fórmula $\gamma \in \Gamma$. Se denota como $M, w \Vdash \Gamma$.

Definición A.1.7. (Consecuencia) Una fórmula ϕ es consecuencia de un conjunto de fórmulas Γ cuando (Fernández et al. 2007):

si $M, w \Vdash \Gamma$ entonces $M, w \Vdash \phi$ (para todo mundo w de todo modelo M).[3]

Consecuencia restringida a un conjunto de modelos La relación de consecuencia propuesta es muy exigente: debe verificarse en todo mundo de todo modelo. De esta manera, $p \nvDash \Diamond p$, ya que existen mundos donde p es verdadera sin que lo sea en ninguno de sus mundos accesibles. No obstante, si sólo se consideraran los modelos basados en marcos reflexivos, en todo mundo en que sea verdadera p debe serlo $\Diamond p$. Es decir, $\Diamond p$ es consecuencia lógica de p (restringido al conjunto de modelos basados en marcos reflexivos).

[3] **Notación:** esta relación semántica entre el conjunto de fórmulas Γ y la fórmula ϕ se denota $\Gamma \Vdash \phi$. Se suele escribir $\psi, \ldots, \chi \Vdash \phi$ en vez de $\{\psi, \ldots, \chi\} \Vdash \phi$, omitiendo las llaves que delimitan las fórmulas del conjunto Γ. Así, se denota $\psi \Vdash \phi$ cuando el conjunto Γ conste de una única fórmula ψ.
La definición requiere algo más que el mero hecho de que las fórmulas Γ y la fórmula 0 coincidan, puntual y ocasionalmente, en ser verdaderas. Se requiere que 'donde quiera que se evalúen (mundo y modelo) no pueda ocurrir que se satisfacen las fórmulas Γ y no se satisfaga ϕ'.

Esta relación restringida se denota como $\Gamma \nvDash_{\mathcal{M}} \phi$ y se define como:

si $M, w \Vdash \Gamma$ entonces $M, w \Vdash \phi$, (para todo mundo w de todo modelo $M \in \mathcal{M}$)

Es importante tener en cuenta que, restringidos a un cierto conjunto de modelos $\mathcal{M}$:

- todas las relaciones generales de consecuencia se siguen manteniendo; por ejemplo, $(\Box(\phi \to \psi) \nvDash_{\mathcal{M}} (\Box\phi \to \Box\psi))$
- se satisfacen relaciones que no se satisfacían en el conjunto total de modelos; por ejemplo, $p \nvDash_{\mathcal{M}} \Diamond p$ (para $\mathcal{M}$ igual al conjunto de modelos sobre marcos reflexivos).

Equivalencia

Definición A.1.8. (Equivalencia) Las fórmulas ϕ y ψ son equivalentes si $\phi \Vdash \psi$ y $\psi \Vdash \phi$. Para representar esta relación semántica entre ambas fórmulas se utiliza la notación $\phi \equiv \psi$ (Fernández et al. 2007).

Nótese que la definición de equivalencia se formalizó utilizando el concepto de consecuencia (A.1.7). De esta definición resulta que dos fórmulas son equivalentes si en todo mundo de todo modelo se satisface una si y sólo si se satisface la otra.

Ejemplo A.1.9. (Equivalencias modales básicas) Entre las equivalencias modales más útiles e inmediatas se encuentran:

- $\neg\Box\neg\phi \equiv \Diamond\phi$
- $\Box(\phi \wedge \psi) \equiv \Box\phi \wedge \Box\psi$
- $\Diamond(\phi \vee \psi) \equiv \Diamond\phi \vee \Diamond\psi$

A.2. Teoría de la correspondencia

Se expuso que, dado un conjunto de modelos $\mathcal{M}$, existe un conjunto de fórmulas válidas en él. Algunas, porque son válidas en todo modelo, junto a otras particularmente válidas en $\mathcal{M}$. Considérese una fórmula ϕ de este segundo grupo: no es válida (en general), pero sí lo es sobre los modelos de $\mathcal{M}$. No se descarta que esta fórmula ϕ sea también válida sobre otros conjuntos de modelos $\mathcal{M}_1, \ldots, \mathcal{M}_k$.

> Cuando no ocurre esto, *cuando una fórmula es válida sobre un conjunto de modelos y sólo sobre ese conjunto*, esta fórmula modal caracteriza sintácticamente a todo ese conjunto (normalmente infinito) de modelos.

Esta sección se restringe al estudio de los conjuntos de modelos basados en marcos y de las fórmulas modales que los caracterizan. Ejemplos de estos conjuntos son: todos los modelo con marco reflexivo, con marco transitivo, con marco transitivo y euclidiano, cuya relación sea una relación de equivalencia o de orden. En general, todas estas propiedades se pueden definir por medio de la lógica clásica de primer (o de segundo) orden.

Esta línea de investigación suele denominarse **Teoría de la Correspondencia** porque trata de buscar la correspondencia adecuada entre las fórmulas modales y las fórmulas en otros lenguajes, cuando todas ellas se satisfacen exclusivamente sobre las mismas estructuras relacionales (Fernández et al. 2007).

En general, los teoremas presentan una estructura parecida a la del siguiente, donde una propiedad de la relación se presenta si y sólo si se satisface una fórmula (o, equivalentemente, un esquema):

Teorema A.2.1. *Sea $F = \langle W, R \rangle$ un marco, entonces (Fernández et al. 2007):*

R es transitiva *si y sólo si* $F \Vdash (\Box p \to \Box\Box p)$ *si y sólo si* $F \Vdash (\Box\phi \to \Box\Box\phi)$

Demostración. • Si R es transitiva entonces $F \Vdash (\Box p \to \Box\Box p)$:

Considérese un modelo M basado en F (con relación transitiva, por hipótesis) y un mundo$w \in W$. Se satisface este condicional $M, w \Vdash (\Box p \to \Box\Box p)$ si y sólo si la suposición $M, w \Vdash \Box p$ conduce a $M, w \Vdash \Box\Box p$. Supongamos entonces$M, w \Vdash \Box p$. Es decir, que para todo w' accesible desde w se satisface p (1).

Considérense todos los mundos w'' relacionados con estos w'. Como la relación R es transitiva, también están relacionados con w. Por el resultado (1) en todos ellos se satisface p. Luego en cada w' se satisface $\Box p$ y en w se satisface $\Box\Box p$.

• Si $F \Vdash (\Box p \to \Box\Box p)$ entonces la relación R de F es transitiva:
Si la fórmula se satisface en F entonces se satisface para todo mundo y toda asignación sobre F. Luego, se satisface para todo mundo dada una asignación particular (que se construirá).

Se intenta demostrar que la satisfacción sobre esta asignación particular implica la transitividad de R. O de manera equivalente, que si R no fuese transitiva, la fórmula no sería satisfacible para esa asignación. De esta manera, se demuestra que cuando es cierto que la fórmula se satisface para todos los modelos de F debe ser transitiva (o fallaría, al menos, en ese modelo).

Se elije un mundo cualquiera t y se considera una asignación $v(p) = w \in W \mid Rtw$; es decir, p es verdadera sólo en los mundos relacionados con t. Esta asignación garantiza (1) que $F, v, t \Vdash \Box p$.

Inicialmente se supuso $F \Vdash (\Box p \to \Box\Box p)$, que junto con (1) implica que $F, v, t \Vdash \Box p$ (2).

De esta manera, para todo w', w'' tales que Rtw' y $Rw'w''$, de (2) se sigue que en w'' se satisface p. Y por lo tanto, que Rtw'', puesto que sólo satisfacían p los mundos relacionados con t.

Observe, del párrafo anterior, que se ha garantizado que para todo w', w'' tales que Rtw' y $Rw'w''$, entonces Rtw''. Es decir, la transitividad de la relación, puesto que no se ha utilizado ninguna propiedad específica del mundo t y por lo tanto el desarrollo es válido para todo t.

• Si $F \Vdash (\Box p \rightarrow \Box\Box p)$ entonces $F \Vdash (\Box\phi \rightarrow \Box\Box\phi)$:

La segunda fórmula es una instancia por sustitución de la primera. Se preserva la validez porque nos hemos ceñido a conjuntos de modelos basados en marcos.

• Si $F \Vdash (\Box\phi \rightarrow \Box\Box\phi)$ entonces $F \Vdash (\Box p \rightarrow \Box\Box p)$:

La segunda fórmula se ajusta al esquema de la primera, simplemente considerando § = p (Fernández et al. 2007). □

El cuadro A.2.1 facilita algunas fórmulas modales junto al tipo de relaciones que caracterizan. Se adjunta también el nombre histórico dado a estas fórmulas (o esquemas de fórmulas).

Reflexiva	$(\Box p \rightarrow p)$	(T)
Simétrica	$(p \rightarrow \Box\Diamond p)$	(B)
Transitiva	$(\Box p \rightarrow \Box\Box p)$	(4)
Euclidiana	$(\Box p \rightarrow \Box\Diamond p)$	(5)
Determinista	$(\Diamond p \rightarrow \Box p)$	(Q)
Serial	$(\Box p \rightarrow \Diamond p)$	(D)
Funcional	$(\Box p \rightarrow \Diamond p)$	

Cuadro A.2.1: Fórmulas modales que caracterizan relaciones binarias

La demostración de estas correspondencias se efectúa de forma análoga a la del teorema A.2.1. El único paso no trivial consiste en demostrar la propiedad dada la satisfacción de la fórmula. Para ello es preciso encontrar una asignación particular afortunada, de la que se siga el resultado.

A.3. Lógicas normales

Cada lógica normal es un conjunto determinado de fórmulas. Como tales conjuntos, si una fórmula pertenece a uno de ellos y no a otro, resultan ser conjuntos distintos: lógicas normales distintas. Existen multitud de lógicas normales distintas. Sin embargo, todas ellas cumplen, para serlo, características comunes.

Definición A.3.1. (Lógica normal) Una lógica normal es un conjunto de fórmulas que incluye (Fernández et al. 2007):

1. todas las tautologías
2. todas las fórmulas de la forma $(\Box(\phi \rightarrow \psi) \rightarrow (\Box\phi \rightarrow \Box\psi))$

y es cerrado respecto de:

3. **modus ponens:** si incluye ϕ y $(\phi \rightarrow \psi)$, entonces incluye ψ
4. **generalización:** si incluye ϕ, entonces incluye $\Box\phi$
5. **sustitución uniforme:** si incluye ϕ, entonces incluye cualquier instancia suya por sustitución uniforme

Observe que esta definición no hace ninguna referencia a estructuras relacionales. Cualquier conjunto de fórmulas que sintácticamente garantice estas propiedades

es una lógica normal. No obstante, esta definición recuerda a los resultados del teorema A.1.6. Efectivamente, toda lógica basada en marcos (todas las fórmulas válidas sobre un conjunto de marcos) resulta ser una lógica normal. Sin embargo, no a toda lógica normal le corresponde una lógica basada en marcos. Aquí no se considera ninguna de estas lógicas normales no basadas en marcos.

Lógica K Es la menor de las lógicas normales incluye sólo las tautologías y la fórmula K, así como las fórmulas que resultan del cierre frente a las tres condiciones propuestas. Todas sus fórmulas son válidas.

Lógica KT Resulta tras añadir el axioma $T := \Box p \rightarrow p$, a la lógica K (junto a todas las fórmulas que se obtienen por cierre). Todas sus fórmulas son válidas sobre marcos reflexivos.

Lógica $K4$ Resulta tras añadir el axioma $4 := \Box p \rightarrow \Box\Box p$, a la lógica K (junto a todas las fórmulas que se obtienen por cierre). Todas sus fórmulas son válidas sobre marcos transitivos.

Lógica $KT4$ **ó** $S4$ Resulta tras añadir T y 4 a la lógica K (considerando todos sus cierres). Todas sus fórmulas son válidas sobre el conjunto de marcos reflexivos y transitivos.

Lógica $KT45$ **ó** $S5$ Resulta tras añadir T, 4 y $5 := \Box p \rightarrow \Box\Diamond p$ a la lógica K (considerando todos sus cierres). Todas sus fórmulas son válidas sobre el conjunto de marcos reflexivos, euclidianos y transitivos, que resultan también ser los marcos reflexivos, simétricos y transitivos, es decir, sobre todas las relaciones que sean de equivalencia.

Existen otras muchas lógicas normales de interés. En los ejemplos que se han citado (y en otros omitidos) estos conjuntos de fórmulas presentan 3 perspectivas complementarias que las hacen especialmente atractivas:

- Por un lado, todas sus fórmulas son válidas sobre conjuntos de marcos que tienen una propiedad formal de interés.
- Así, cuando se quiere modelar un concepto o un sistema (tiempo, agentes u otros) se escoge la lógica adecuada a las propiedades que presentan las relaciones del sistema (simetría, transitividad entre otras).
- Por otro lado, todas estas lógicas tienen diversos sistemas de demostración bien definidos. Por ejemplo, a partir de los axiomas K, T y 4, mediante reglas de inferencia adecuadas, se puede ir generando sintácticamente fórmulas de $KT4$, de las que se garantiza que son válidas en su conjunto de marcos (reflexivos y transitivos).

Apéndice B

Detalles de implementación de *chronos-tmw*

Al iniciar *chronos-tmw* se presenta una pantalla como la que se encuentra en la figura B.1 siguiente:

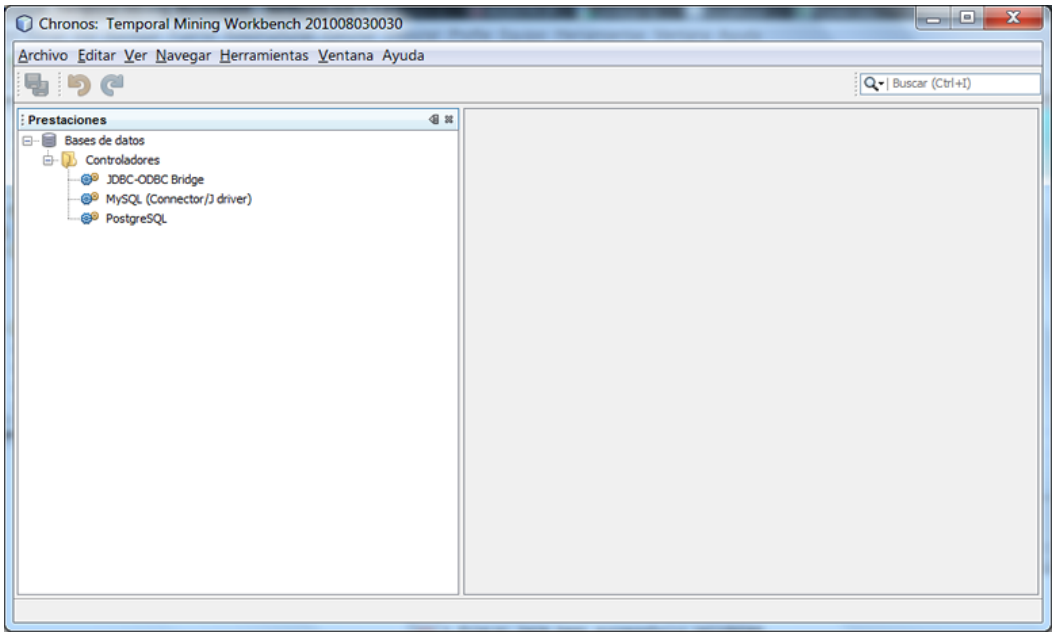

Figura B.1: Pantalla inicial de *chronos-tmw*.

Tal como se puede ver en dicha pantalla se despliegan una lista de controladores. El único controlador que permitirá la funcionalidad de descubrimiento de patrones secuenciales es el controlador de bases de datos *JDBC-ODBC Bridge*. Cada controlador tiene un menú emergente con el que se muestra en la figura .

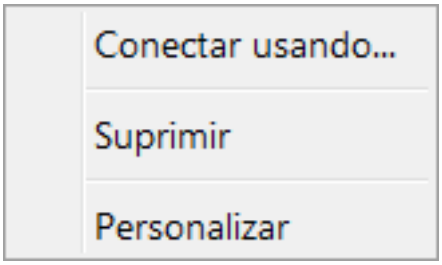

Figura B.2: Menú emergente de los controladores de bases de datos.

La funcionalidad que se debe utilizar en ese menú es "Conectar usando...". Esto va a desplegar un diálogo (fig. B.3) que permite realizar una conexión *JDBC-ODBC* Bridge hacia la base que el usuario escoja.

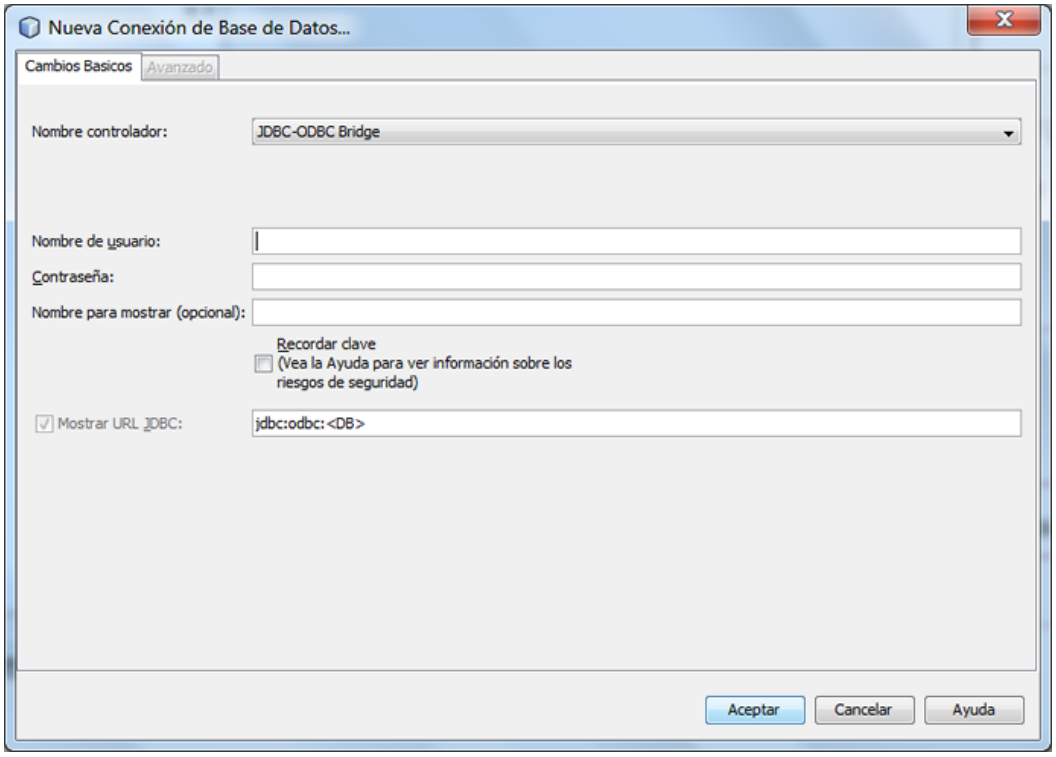

Figura B.3: Diálogo de configuración de conexión de base de datos.

En dicho diálogo es necesario indicar el nombre de usuario y la contraseña. También se puede indicar un nombre opcional para la conexión. <DB> se debe reemplazar por el nombre del Nombre de la Fuente de Datos ODBC (DSN ODBC, por sus siglas en inglés).

Una vez conectada la base de datos, esto permite explorar la base de datos de la conexión y se despliegan las demás bases de datos a las cuales el usuario tiene acceso. La base de datos cuyo nombre está en negrita es la base de datos escogida por omisión. Cada nombre de base de datos tiene un menú emergente que permite establecerla como base de datos por omisión. El ícono de la conexión y su nombre tienen también un menú emergente que se muestra a continuación en la figura B.4:

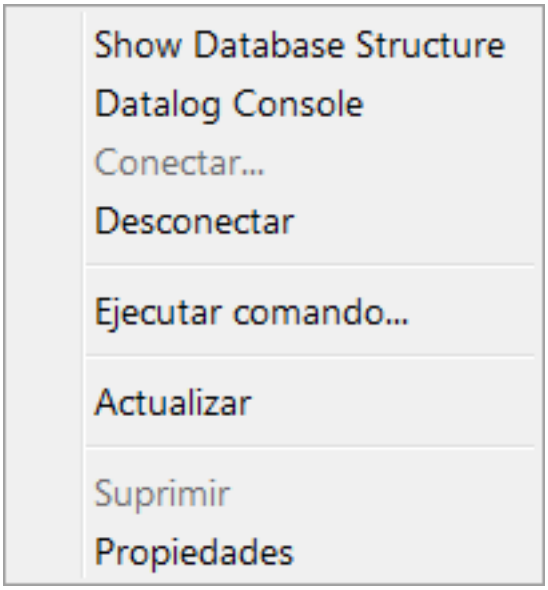

Figura B.4: Menú emergente de la conexión a base de datos.

Para *chronos-tmw* se programaron las funcionalidades "Show Database Structure" y "Datalog Console". Estás dos funcionalidades se agregan al menú heredando del objeto *BaseAction* de *NetBeans*. Las demás funcionalidades provienen de la plataforma *NetBeans*. "Show Database Structure" programado en un módulo *NetBeans* muestra de manera gráfica las tablas en la base de datos por omisión de la conexión. En un futuro se podría proveer a través de esto una minería de datos más gráfica tal como se plantea en el posible trabajo futuro (capítulo 5.2). La funcionalidad "Datalog Console" abre una consola en donde se pueden hacer consultas al motor de programación lógica *XSB*. La consola es una clase final llamada *Datalog-TopComponent* que hereda de la clase *TopComponent* de *Netbeans* y que implementa la interfaz *PrologOutputListener* de *Interprolog*. Esta consola se muestra en la figura a continuación:

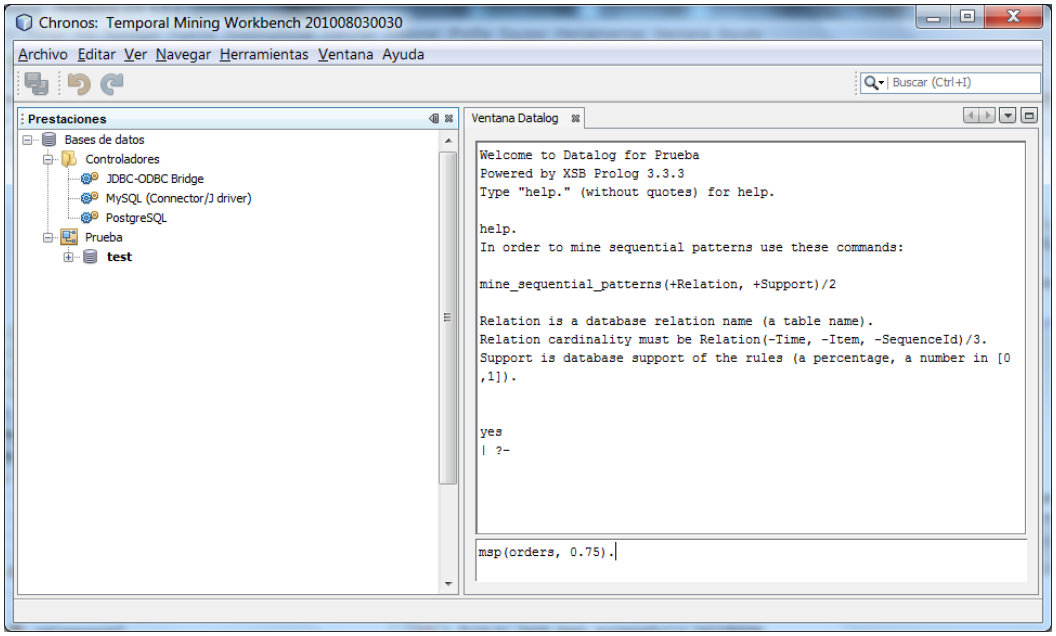

Figura B.5: Consola de programación lógica en *chronos-tmw*.

Cuando se muestra la consola se llama a la acción *componentShowing*. Cuando se cierra la consola se llama a la acción *componentHidden*. Cuando se llaman a las acciones *componentShowing* y *componentHidden* de la clase *DatalogTopComponent* se crea el motor de programación lógica y se destruye el motor de programación lógica respectivamente. Además, en *componentShowing* se le pasa a XSB la información necesaria para que pueda acceder la base de datos.

Cuando se crea el motor de programación lógica, luego de pasar los datos de conexión a la base de datos y de conectar las tablas de la base de datos, se envía al motor de programación lógica la instrucción de consultar un archivo llamado *sequential_patterns.P* que contiene el programa lógico que permite el descubrimiento de patrones secuenciales utilizando lógica temporal.

Apéndice C

Programa para truncar patrones no maximales

El programa siguiente es un programa *Perl* que se utilizó, sobre la salida del algoritmo *Patrones Secuenciales Generalizados* de *RapidMiner*, para eliminar los patrones que se encuentra contenidos en otros patrones. Se utilizó *Perl* dado que como lenguaje es muy útil para la manipulación de texto o reportes y como los resultados de *RapidMiner* son un reporte de texto se hizo obvia su utilización. El objetivo del programa fue limpiar los datos, no se buscó eficiencia.

```
1 # Los siguientes módulos deben estar instalados – todos están disponibles en CPAN
2 use 5.12.0;
3 use strict;
4 use Carp ();
5 use FileHandle;
6
7 print "GSP 45%\n\n";
8
9 my @patterns45 = ();
10 my @support45 = ();
11
12 my $pat45 = new FileHandle;
13 $pat45->open("<45.pat") or Carp::confess "Can't open 45.pat file for reading!";
14
15 while (my $line = $pat45->getline) {
16     my @pattern = ();
17     while ($line =~ /(<(item\s=\s\d+(,\s)*)+>)/g) {
18         my @itemset = ();
19         my $items = $1;
20         while ($items =~ /item\s=\s(\d+)(,\s)*/g) {
21             push @itemset, $1;
```

```
22          }
23          push @pattern, [ @itemset ];
24      }
25      push @patterns45, [ @pattern ];
26      if ($line =~ /^(0.\d+):/) {
27          push @support45, $1;
28      }
29  }
30
31  for (my $k = 5; $k > 1; $k--) {
32      for (my $pattern = 0; $pattern < scalar @patterns45; $pattern++) {
33          if (defined($patterns45[$pattern])) {
34              if (scalar(@{$patterns45[$pattern]}) == $k) {
35                  for (my $compare = 0; $compare < scalar @patterns45; $compare++) {
36                      if (($compare != $pattern) && defined($patterns45[$compare]) && (scalar
                            (@{$patterns45[$compare]})<=$k)) {
37                          if (is_subpattern($patterns45[$compare], $patterns45[$pattern])) {
38                              delete $patterns45[$compare];
39                          }
40                      }
41                  }
42              }
43          }
44      }
45  }
46
47  while (my ($support, $pattern) = each @patterns45) {
48      if (defined($pattern)) {
49          print $support45[$support].': ';
50          foreach my $itemset (@{$pattern}) {
51              print '<';
52              while (my ($index, $item) = each @{$itemset}) {
53                  print "item = $item";
54                  if ($index != (scalar(@{$itemset})-1)) {
55                      print ', ';
56                  }
57              }
58              print '> ';
59          }
60          print "\n";
61      }
62  }
63
64  print "\n\nGSP 60%\n\n";
65
66  my @patterns60 = ();
67  my @support60 = ();
68
69  my $pat60 = new FileHandle;
70  $pat60->open("<60.pat") or Carp::confess "Can't open 60.pat file for reading!";
71
72  while (my $line = $pat60->getline) {
73      my @pattern = ();
74      while ($line =~ /(<(item\s=\s\d+(,\s)*)+>)/g) {
75          my @itemset = ();
76          my $items = $1;
77          while ($items =~ /item\s=\s(\d+)(,\s)*/g) {
78              push @itemset, $1;
79          }
80          push @pattern, [ @itemset ];
81      }
82      push @patterns60, [ @pattern ];
83      if ($line =~ /^(0.\d+):/) {
84          push @support60, $1;
85      }
86  }
87
```

```
88 for (my $k = 5; $k > 1; $k--) {
89     for (my $pattern = 0; $pattern < scalar @patterns60; $pattern++) {
90         if (defined(@{$patterns60[$pattern]})) {
91             if (scalar(@{$patterns60[$pattern]}) == $k) {
92                 for (my $compare = 0; $compare < scalar @patterns60; $compare++) {
93                     if (($compare != $pattern) && defined($patterns60[$compare]) && (scalar
                           (@{$patterns60[$compare]})<=$k)) {
94                         if (is_subpattern($patterns60[$compare], $patterns60[$pattern])) {
95                             delete $patterns60[$compare];
96                         }
97                     }
98                 }
99             }
100        }
101    }
102 }
103
104 while (my ($support, $pattern) = each @patterns60) {
105    if (defined($pattern)) {
106        print $support60[$support].': ';
107        foreach my $itemset (@{$pattern}) {
108            print '<';
109            while (my ($index, $item) = each @{$itemset}) {
110                print "item = $item";
111                if ($index != (scalar(@{$itemset})-1)) {
112                    print ', ';
113                }
114            }
115            print '> ';
116        }
117        print "\n";
118    }
119 }
120
121 #
122 # Subroutines
123 #
124
125 sub is_subpattern {
126    my $comp = shift;
127    my @compare = @{$comp};
128    my $pat = shift;
129    my @pattern = @{$pat};
130
131    for (my $i=0; $i < scalar(@compare); $i++) {
132        for (my $j=0; $j < scalar(@{$compare[$i]}); $j++) {
133            if (!($compare[$i][$j] ~~ @{$pattern[$i]})) {
134                return 0;
135            }
136        }
137    }
138    return 1;
139 }
```

Apéndice D

Patrones encontrados por *RapidMiner* en los datos generados por KNIME y con un 60 % de apoyo de los datos

Los siguientes patrones son los patrones encontrados por *RapidMiner* en los datos generados por KNIME. Estos patrones tienen un apoyo de los datos mayor o igual a 0.6. Además, son maximales, i.e., no están contenidos dentro de ningún otro patrón.

0.620: <item = 3> <item = 3>

0.625: <item = 3> <item = 22>

0.620: <item = 3> <item = 23>

0.615: <item = 3> <item = 43>

0.615: <item = 3> <item = 70>

0.620: <item = 3> <item = 84>

0.610: <item = 3> <item = 90>

0.655: <item = 3> <item = 92>

0.670: <item = 3> <item = 152>

0.625: <item = 22> <item = 3>

0.605: <item = 22> <item = 23>

0.600: <item = 22> <item = 43>

0.625: <item = 22> <item = 84>

0.620: <item = 22> <item = 92>

0.660: <item = 22> <item = 152>

0.695: <item = 23> <item = 3>

0.635: <item = 23> <item = 22>

0.655: <item = 23> <item = 23>

0.620: <item = 23> <item = 34>

0.695: <item = 23> <item = 43>

0.635: <item = 23> <item = 70>

0.645: <item = 23> <item = 84>

0.660: <item = 23> <item = 90>

0.715: <item = 23> <item = 92>

0.605: <item = 34> <item = 3>

0.615: <item = 34> <item = 23>

0.615: <item = 34> <item = 43>

0.615: <item = 34> <item = 84>

0.625: <item = 34> <item = 92>

0.630: <item = 34> <item = 152>

0.710: <item = 43> <item = 3>

0.660: <item = 43> <item = 22>

0.730: <item = 43> <item = 23>

0.640: <item = 43> <item = 34>

0.700: <item = 43> <item = 43>

0.650: <item = 43> <item = 70>

0.680: <item = 43> <item = 84>

0.645: <item = 43> <item = 90>

0.690: <item = 45, item = 127>

0.625: <item = 70> <item = 3>

0.640: <item = 70> <item = 23>

0.645: <item = 70> <item = 43>

0.620: <item = 70> <item = 84>

0.610: <item = 70> <item = 90>

0.675: <item = 70> <item = 92>

0.685: <item = 70> <item = 152>

0.625: <item = 84> <item = 3>

0.620: <item = 84> <item = 43>

0.625: <item = 84> <item = 92>

0.680: <item = 84> <item = 152>

0.645: <item = 90> <item = 3>

0.600: <item = 90> <item = 22>

0.645: <item = 90> <item = 23>

0.605: <item = 90> <item = 34>

0.660: <item = 90> <item = 43>

0.610: <item = 90> <item = 70>

0.645: <item = 90> <item = 84>

0.620: <item = 90> <item = 90>

0.665: <item = 90> <item = 92>

0.675: <item = 90> <item = 152>

0.705: <item = 92> <item = 3>

0.635: <item = 92> <item = 22>

0.690: <item = 92> <item = 23>

0.685: <item = 92> <item = 34>

0.700: <item = 92> <item = 43>

0.600: <item = 92> <item = 70>

0.665: <item = 92> <item = 84>

0.645: <item = 92> <item = 90>

0.725: <item = 92> <item = 92>

0.745: <item = 92> <item = 152>

0.720: <item = 152> <item = 3>

0.690: <item = 152> <item = 22>

0.690: <item = 152> <item = 34>

0.665: <item = 152> <item = 70>

0.675: <item = 152> <item = 84>

0.690: <item = 152> <item = 90>

0.600: <item = 23> <item = 152> <item = 152>

0.600: <item = 43> <item = 92> <item = 152>

0.610: <item = 43> <item = 152> <item = 152>

0.600: <item = 152> <item = 23> <item = 92>

0.600: <item = 152> <item = 43> <item = 92>

0.600: <item = 152> <item = 43> <item = 152>

0.615: <item = 152> <item = 92> <item = 92>

0.620: <item = 152> <item = 92> <item = 152>

0.610: <item = 152> <item = 152> <item = 3>

0.610: <item = 152> <item = 152> <item = 23>

0.600: <item = 152> <item = 152> <item = 43>

0.620: <item = 152> <item = 152> <item = 92>

0.635: <item = 152> <item = 152> <item = 152>

0.615: <item = 3> <item = 122, item = 123, item = 124>

0.600: <item = 23> <item = 122, item = 123, item = 124>

0.610: <item = 43> <item = 122, item = 123, item = 124>

0.650: <item = 92> <item = 122, item = 123, item = 124>

0.640: <item = 122, item = 123, item = 124> <item = 3>

0.615: <item = 122, item = 123, item = 124> <item = 23>

0.605: <item = 122, item = 123, item = 124> <item = 34>

0.620: <item = 122, item = 123, item = 124> <item = 43>

0.615: <item = 122, item = 123, item = 124> <item = 70>

0.605: <item = 122, item = 123, item = 124> <item = 84>

0.615: <item = 122, item = 123, item = 124> <item = 90>

0.620: <item = 122, item = 123, item = 124> <item = 92>

0.675: <item = 122, item = 123, item = 124> <item = 152>

0.665: <item = 152> <item = 122, item = 123, item = 124>

Apéndice E

Patrones encontrados por *chronos-tmw* en los datos generados por KNIME y con un 60 % de apoyo de los datos

Los patrones a continuación fueron encontrados por *chronos-tmw* en los datos generados por KNIME. Estos patrones tienen un apoyo mayor o igual a 0.6 en los datos. Además, estos patrones son maximales.

```
+-----------------------------------------------------+
| Sequential Patterns Mining with Temporal Logic |
+-----------------------------------------------------+

Support: 0.6000
Creating sequence database, please wait...
Time creating sequence database: %5.772 CPU in 6.086 seconds (94% CPU)

Sequence database: 200 sequences
Creating large-itemsets (1-itemsets), please wait...
Time creating large-itemsets: %340.894 CPU in 346.257 seconds (98% CPU)

Large Itemsets with more than 120.0000 sequences support: 58
Finding patterns, please wait...
Time finding patterns: %386.477 CPU in 393.037 seconds (98% CPU)

Patterns:
0.6350: ([152] and f [152]) and f [152]
```

```
0.6200: ([152] and f [152]) and f [92]
0.6000: ([152] and f [152]) and f [43]
0.6100: ([152] and f [152]) and f [23]
0.6100: ([152] and f [152]) and f [3]
0.6200: ([152] and f [92]) and f [152]
0.6150: ([152] and f [92]) and f [92]
0.6000: ([152] and f [43]) and f [152]
0.6000: ([152] and f [43]) and f [92]
0.6000: ([152] and f [23]) and f [92]
0.6100: ([43] and f [152]) and f [152]
0.6000: ([43] and f [92]) and f [152]
0.6000: ([23] and f [152]) and f [152]
0.6200: [3] and f [3]
0.6150: [3] and f [70]
0.6200: [3] and f [84]
0.6250: [3] and f [22]
0.6200: [3] and f [23]
0.6700: [3] and f [152]
0.6100: [3] and f [90]
0.6550: [3] and f [92]
0.6150: [3] and f [43]
0.6150: [3] and f [122,123,124]
0.6850: [70] and f [152]
0.6750: [70] and f [92]
0.6100: [70] and f [90]
0.6200: [70] and f [84]
0.6450: [70] and f [43]
0.6400: [70] and f [23]
0.6250: [70] and f [3]
0.6800: [84] and f [152]
0.6250: [84] and f [92]
0.6200: [84] and f [43]
0.6250: [84] and f [3]
0.6600: [22] and f [152]
0.6200: [22] and f [92]
0.6250: [22] and f [84]
0.6000: [22] and f [43]
0.6050: [22] and f [23]
0.6250: [22] and f [3]
0.6950: [23] and f [3]
0.6350: [23] and f [70]
0.6450: [23] and f [84]
0.6350: [23] and f [22]
0.6550: [23] and f [23]
0.6600: [23] and f [90]
0.7150: [23] and f [92]
0.6200: [23] and f [34]
0.6950: [23] and f [43]
0.6000: [23] and f [122,123,124]
0.7200: [152] and f [3]
0.6650: [152] and f [70]
0.6750: [152] and f [84]
0.6900: [152] and f [22]
0.6900: [152] and f [90]
0.6900: [152] and f [34]
0.6650: [152] and f [122,123,124]
0.6450: [90] and f [3]
0.6100: [90] and f [70]
0.6450: [90] and f [84]
0.6000: [90] and f [22]
0.6450: [90] and f [23]
0.6750: [90] and f [152]
0.6200: [90] and f [90]
0.6650: [90] and f [92]
0.6050: [90] and f [34]
0.6600: [90] and f [43]
0.7050: [92] and f [3]
```

```
0.6000: [92] and f [70]
0.6650: [92] and f [84]
0.6350: [92] and f [22]
0.6900: [92] and f [23]
0.7450: [92] and f [152]
0.6450: [92] and f [90]
0.7250: [92] and f [92]
0.6850: [92] and f [34]
0.7000: [92] and f [43]
0.6500: [92] and f [122,123,124]
0.6300: [34] and f [152]
0.6250: [34] and f [92]
0.6150: [34] and f [84]
0.6150: [34] and f [43]
0.6150: [34] and f [23]
0.6050: [34] and f [3]
0.7100: [43] and f [3]
0.6500: [43] and f [70]
0.6800: [43] and f [84]
0.6600: [43] and f [22]
0.7300: [43] and f [23]
0.6450: [43] and f [90]
0.6400: [43] and f [34]
0.7000: [43] and f [43]
0.6100: [43] and f [122,123,124]
0.6400: [122,123,124] and f [3]
0.6150: [122,123,124] and f [70]
0.6050: [122,123,124] and f [84]
0.6150: [122,123,124] and f [23]
0.6750: [122,123,124] and f [152]
0.6150: [122,123,124] and f [90]
0.6200: [122,123,124] and f [92]
0.6050: [122,123,124] and f [34]
0.6200: [122,123,124] and f [43]
Total patterns: 102
```

Apéndice F

Cambios realizados al código de Quest

Quest fue realizado en versiones antiguas de compiladores de C++. Los cambios fueron básicamente para poder compilar el programa en versiones más modernas. La compilación correcta no hace necesariamente al programa semánticamente correcto.

Se hicieron tres tipos de cambios:

- Cambios en las extensiones. Se pasaron las extensiones .c a .cpp para que los nombres de archivo sean más estándar.
- Cambios en el código para arreglar errores e incompatibilidades. Uno de los cambios más inquietantes se dio en el archivo gen.cpp. Este cambio fue para evitar acceder a memoria no asignada.
- Se agregaron archivos nuevos para construir el proyecto en compiladores diferentes.

Archivo	**Diferencias en formato *diff*.**
dist.cpp	2c2,5 < #include <values.h> --- > #include <limits.h> > > #define MAXLONG LONG_MAX >
dist.h	5c5 < float ran0(long &idum);
gen.cpp	1a2 > #include <string> 3d3 < #include <string.h> 5a6 > using namespace std; 891c892 < if (trans[i]->size() > 0) --- > if ((trans[i]!=NULL) && (trans[i]->size() > 0)) 899c900 < if (trans[i]->size() > 0) --- > if ((trans[i]!=NULL) && (trans[i]->size() > 0))
gen.h	3,4d2 < #include <stream.h> < #include <fstream.h> 5a4,8 > #include <iostream> > #include <iomanip> > #include <fstream> > > using namespace std;

Archivo	**Diferencias en formato *diff*.**

main.c	`1,3c1,7`
	`< #include <fstream.h>`
	`< #include <stdio.h>`
	`< #include <new.h>`
	`---`
	`> #include <cstdlib>`
	`> #include <iostream>`
	`> #include <fstream>`
	`> #include <new>`
	`>`
	`> using namespace std;`
	`>`

poidev.cpp	`1a2`
	`> #include "gammln.h"`
	`10d10`
	`< float gammln(float xx);`