Universidad de Granada
Escuela Técnica Superior de Ingeniería Informática
y Telecomunicaciones
Department of Computer Science and Artificial
Intelligence (DECSAI)
Intelligent Databases and Information Systems
(IDBIS)

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Telecommunicatie en
Informatieverwerking (TELIN)
Onderzoeksgroep Database, Document en Content
Management (DDCM)

Fuzzy Temporal Information Treatment in Relational Databases

Programa de doctorado: Tecnologías de la Información y la Comunicación

---

José Enrique Pons Frías

Dissertation submitted in accordance with the
requirements for the double degree of
Doctor of Computer Science Engineering by
Ghent University and
Doctor by University of Granada
Academic year 2012-2013

Universidad de Granada
Escuela Técnica Superior de Ingeniería Informática
y Telecomunicaciones
Department of Computer Science and Artificial
Intelligence (DECSAI)
Intelligent Databases and Information Systems
(IDBIS)

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Telecommunicatie en
Informatieverwerking (TELIN)
Onderzoeksgroep Database, Document en Content
Management (DDCM)

Promoters: Prof. Dr. Olga Pons Capote [1]
Prof. Dr. Guy de Tré [2]

[1] Universidad de Granada
Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Department of Computer Science and Artificial Intelligence
C/ Periodista Daniel Saucedo Aranda s/n, 18071 Granada, España

Tel.: +34-958-24.40.19
Fax.: +34-958-24.33.17

[2] Universiteit Gent
Faculteit Ingenieurswetenschappen

Vakgroep Telecommunicatie en Informatieverwerking
St-Pietersnieuwstraat 41, B-9000 Gent, Belgium

Tel.: +32-9-264.34.12
Fax.: +32-9-264.42.95

This research has been supported by the grant BES-2009-013805 within the research project TIN2008-02066: *Fuzzy Temporal Information treatment in relational DBMS*.

Dissertation submitted in accordance with the
requirements for the double degree of
Doctor of Computer Science Engineering by
Ghent University and
Doctor by University of Granada
Academic year 2012-2013

El doctorando José Enrique Pons Frías y los directores de la tesis Olga Pons Capote y Guy de Tré, garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, Mayo de 2013

Los directores de la tesis: El doctorando:

Prof. Dr. Olga Pons Capote Prof. Dr. Guy de Tré José Enrique Pons

Agradecimientos / Dankwoord

Este trabajo no hubiera sido posible sin el apoyo, la ayuda y el cariño de mucha gente.

Primero me gustaría dar las gracias a Olga, mi tutora. Ella me dió la oportunidad de realizar este trabajo. Durante todos estos años me ha apoyado y guiado. También me ha dado libertad para desarrollar mis ideas y comprensión. Su confianza en mi trabajo ha hecho que me esfuerce al máximo. A ella, muchas gracias por estos años.

En segundo lugar, quiero darle las gracias a Ignacio Blanco, Nacho. Él me ha ayudado mucho, en especial los dos primeros años. Siempre encontraba un hueco para atenderme, a pesar de sus maratonianos días de trabajo. Ha sido un amigo, ha mirado más por mí que por sus propios intereses y por ello le doy las gracias.

Los primeros dos años de este período, los pasé en el edificio Mecenas. Allí conocí a Rocío y Coral. Gracias a ellas, las jornadas de trabajo se hacían más llevaderas.

Quiero agradecer también a Guy por su ofrecimiento y disposición. También le quiero agradecer el tiempo que he pasado en su laboratorio de estancia. He aprendido mucho con él y he sido muy feliz levantándome cada mañana para ir a trabajar a su laboratorio. Él, junto con Antoon, Christophe, Daan, Tom y Joachim han sido mi familia en Bélgica. Quiero darle especialmente las gracias a Christophe y Helena, en especial por alojarme en su casa y tratarme con amabilidad y cariño.

Durante estos años he trabajado y colaborado también con otros profesores de la universidad como Nicolás, Amparo, Juan Miguel. Siempre han tratado de echarme una mano.

También quiero darle las gracias a la beca de investigación del ministerio, sin la que este trabajo no hubiera sido posible.

Muchas gracias a mis padres. A mi madre Ana, a mi padre José por todos los sacrificios que han hecho por mí. A mi hermana Eva, por alegrarme.

Gracias también a la familia de Laura por su cariño y comprensión. A sus padres José y Chari por acogerme como uno más. También a Pepe y Francis por ser tan amigos. Con especial cariño le escribo estas líneas a David, quien no se explica que semana tras semana los deberes que me mandan "del cole" nunca se acaben. Gracias por tu amistad incondicional. Alejandro no es capaz de pronunciar mi nombre todavía, así que cada semana me llama de una manera. Nos alegras la vida a todos.

Gracias a todos mis amigos. A Araceli, Juanje, Jesús, Pitu, Nono, Arturo por todos los buenos momentos. También gracias a mis amigos Fergu, Manu, Sara, Quique, Fran por los buenos momentos y el día a día en el CITIC. En mi corazón también están Alberto, Estíbaliz, Diego y Rubén. Hemos disfrutado, trabajado y sufrido mucho, pero ha merecido la pena. Gracias también a Juampa. No puedo expresar con palabras lo mucho que significa nuestra amistad.

A Laura, muchas gracias por tu amor, apoyo, cariño, comprensión y complicidad.

Recibo de ti más de lo que pudiera imaginar. Haces que mi vida sea mejor y me complementas.

A todos vosotros, gracias.

Granada, Mayo de 2013
José Enrique Pons Frías

Table of Contents

List of Tables

List of Figures

List of Acronyms

A

A	After
AD	Anno Domini
AFS	Attanassov (Intuitionistic) Fuzzy Set
API	Application Programming Interface
AR	Allen Relation
AST	Abstract Syntax Tree

B

B	Before
BC	Before Christ
BSD	Bipolar Satisfaction Degree

C

C	Contains
CDEG	Compatibility Degree
CJD	Chronological Julian Date
CORBA	Common Object Request Broker Architecture
CRZ	Crisp Relational Zone
CRUD	CReate, Update and Delete
CTI	Crisp Time Interval

D

D	During
DBMS	Database Management System
DDL	Data Definition Language
DJD	Dublin Julian Date
DML	Data Manipulation Language
DT	Decision Time

E

EJB	Enterprise Java Beans

F

FB	From the Beginning
FEQ	Fuzzy Equal To
FGT	Fuzzy Greater Than
FGEQ	Fuzzy Greater Than of Equal To
FIRST	Fuzzy Interface for Relational Systems
FLEQ	Fuzzy Less Than or Equal To
FLT	Fuzzy Less Than
FRDBMS	Fuzzy Relational Data Base Management System
FSQL	Fuzzy Structured Query Language
FVP	Fuzzy Validity Period
FS	Fuzzy Set
FKRO	Fuzzy Knowledge Representation Ontology

G

GEFRED	Generalized Model for Fuzzy Relational Database

H

HQL	Hibernate Query Language

I

IFS	Atanassov (Intuitionistic) Fuzzy Set
IKC	Ill-known Constraint
IKI	Ill-known Interval
IKTP	Ill-known Time Point
IKTI	Ill-known Time Interval
IKV	Ill-known Value
IVFS	Interval-valued Fuzzy Set

J

JD	Julian Date
JDBC	Java Database Connectivity
JDN	Julian Day Number
JED	Epheris Time
JTA	Java Transaction Api

M

MGT	Much Greater Than
MJD	Modified Julian Day
MLT	Much Less Than

N

NFEQ	Necessity Fuzzy Equal To
NFGT	Necessity Fuzzy Greater Than
NFGEQ	Necessity Fuzzy Greater Than or Equal To
NFLEQ	Necessity Fuzzy Less Than or Equal To
NFLT	Necessity Fuzzy Less Than

O

O	Overlaps
OB	Overlapped By
OMG	Object Management Group
OWA	Order Weighted Averaging

P

PE	Possibly Equals
PF	Possibly Finishes
PFB	Possibly Finished By
PM	Possibly Meets
PMB	Possibly Meets By
POJO	Plain Old Java Object
PK	Primary Key
PS	Possibly Starts
PSB	Possibly Started By
PVP	Possibilistic Valid-time Period

Q

QBC	Query By Criteria
QBE	Query By Example

R

RDBMS	Relational Data Base Management System
RJD	Reduced Julian Day

S

SQL	Structured Query Language

T

TT	Terrestrial Time or Transaction Time
TJD	Truncated Julian Date
TSQL	Temporal Structured Query Language
TDB	Temporal Data Base
THOLD	Threshold
TFS	Two-fold Fuzzy Sets
TM	Triangular Model

U

UB	Uncertain Beginning
UE	Uncertain Ending
UT	Universal Time
UDT	User Defined Time
UML	Unified Modelling Language
UC	Until Changed
UTI	Uncertain Time Indication
UIZ	Uncertain Interval Zone
URZ	Uncertain Relational Zone

V

VT	Valid Time
VST	Valid-time Satisfaction Degree
VID	Version Identifier

W

WWW	World Wide Web

X

XML	Structured Markup Language

Nederlandstalige samenvatting "Behandeling van Vage Temporele Informatie in Relationele Databanken" –Summary in Dutch–

Bij informatiesystemen heerst een bijzondere belangstelling voor het opslaan en behandelen van tijdsafhankelijke gegevens of gegevens met een temporele component (voor de eenvoud zullen we vanaf nu naar deze verwijzen als temporele gegevens). Bij het omgaan met temporele gegevens in een databank is het noodzakelijk om het standaardgedrag van de databank-motor te modificeren. Bij het omgaan met temporele gegevens worden gewoonlijk verschillende versies van dezelfde gegevens opgeslagen. Met andere woorden, de evolutie van de gegevens doorheen de tijd wordt bewaard. Daarom zou een mechanisme moeten worden voorzien om de verschillende versies van de bewaarde gegevens op te slaan en met deze verschillende versies om te gaan en om de consistentie tussen deze verschillende versies te verzekeren.

Daarnaast is de beschikbare temporele informatie gewoonlijk niet perfect. Daarom is het noodzakelijk om een formeel werktuig te voorzien om met de imperfecties in temporele informatie om te gaan. De studies van tijd in taal en in kennis leiden ons tot de conclusie dat mensen op een onzekere, onnauwkeurige en/of vage manier omspringen met tijd. In deze thesis stellen we een formeel model voor om om te gaan met imperfecte tijdsintervallen, dat gebaseerd is op possibiliteitstheorie en vaagverzamelingenleer. Deze beide theorieën voorzien goed gekende formele werktuigen om om te gaan met onzekerheid, onnauwkeurigheid en vaagheid.

In dit werk bestuderen we de behandeling van onnauwkeurige temporele gegevens in een databank. Hiertoe stellen we een theoretisch model voor vage temporele relationele databanken voor. Dit model representeert en behandelt imperfecte temporele gegevens op een consistente manier. De belangrijkste bijdrage van dit model is een benadering van de behandeling van imperfecte temporele informatie die dichter ligt bij de menselijke manier van redeneren.

Het voorgestelde model is compleet. We voorzien de datatypes, de integriteitsbeperkingen, de datadefinitietaal (DDL) en de datamanipulatietaal (DML). Het model lost de belangrijkste problemen op bij het omgaan met temporele informatie in een databank. Het is mogelijk om verscheidene versies voor dezelfde gegevens te bewaren. Het consistentiemechanisme wordt verschaft door middel van de datamanipulatietaal (DML), die wordt geherdefinieerd om de consistentie van de temporele gegevens te verzekeren. Op deze manier stellen we de databankconsistentie veilig, zelfs in de aanwezigheid van imperfecte temporele informatie. In deze thesis bestuderen we het flexibel bevragen van temporele gegevens. Eerst worden vergelijkingsoperatoren uitgebreid door het gebruik van specifieke temporele operatoren (before, after, during, ..., etc.).

Gewoonlijk wordt een booleaanse waarde verkregen als resultaat van een evaluatie. Daarentegen worden, in het possibilistisch raamwerk, possibiliteits- en necessiteitsgraden in het eenheidsinterval [0, 1] verkregen als resultaat van een evaluatie. Deze graden voorzien de gebruiker van meer informatie dan een booleaanse waarde.

Om een krachtiger werktuig te voorzien om de gebruikerspreferenties voor te stellen, wordt bipolaire databankbevraging voorgesteld. Er zijn twee belangrijke benaderingen in de literatuur. Volgens de eerste (genaamd de restrictie-wens benadering) specifieert de gebruiker een verzameling restricties die van kracht moeten zijn voor de geselecteerde objecten. De gebruiker mag ook een verzameling eigenschappen aanleveren die tevens wenselijk zijn voor de geselecteerde objecten. Bijvoorbeeld: een gebruiker kan een wagen willen die donker zou moeten zijn, en het liefst zwart is. Volgens de tweede benadering (genaamd de tevredenheid-ontevredenheid benadering) mag de gebruiker een verzameling beperkingen specifiëren die van kracht moeten zijn voor de geselecteerde objecten en een andere verzameling beperkingen die niet van kracht mogen zijn voor de geselecteerde objecten. Bijvoorbeeld: een gebruiker kan een wagen willen die donker zou moeten zijn, maar zeker niet blauw zou mogen zijn. In deze thesis gebruiken we de tevredenheid-ontevredenheid benadering om temporele databanken op een bipolaire manier te bevragen. Dit is nuttig gebleken in zowel historische als criminologische databanken.

Wanneer een bevraging verwerkt wordt bij toepassing op een databank, moet het verwerkingssysteem enkele berekeningen maken om te bepalen of een object de vereisten in de bevraging vervult of niet. Het belangrijkste probleem is hier de aggregatie van de evaluatie van niet-temporele beperkingen en de evaluatie van temporele beperkingen. In het geval van bipolaire bevraging moeten we bovendien de evaluaties van zowel restricties als wensen of de evaluaties van zowel tevredenheid als ontevredenheid aggregeren. Een van de belangrijkste problemen bij het tonen van de resultaten van een bevraging aan de gebruiker is het rangschikken van deze resultaten. De voor de gebruiker interessantere resultaten zouden een hogere score gegeven moeten worden dan de voor de gebruiker minder interessante resultaten. Daarom voorzien we in dit werk verscheidene methoden voor de aggregatie van de evaluaties van temporele en niet-temporele vereisten. Daarnaast voorzien we enkele methoden voor het rangschikken van de resultaten van een bevraging.

Gewoonlijk wordt een grote hoeveelheid gegevens verkregen als resultaat van een bevraging. Tegenwoordig voorzien de meeste moderne databankbeheersystemen werktuigen om deze informatie samen te vatten en om de resultaten van een bevraging op een gebruiksvriendelijke manier te presenteren. Bij het bevragen van temporele gegevens is het van bijzonder belang om de geldigheidsperiode van de opgevraagde gegevens te tonen. Gewoonlijk wordt deze temporele informatie gevisualiseerd als een interval volgens een zekere tijdslijn. Desalniettemin is deze voorstelling niet conveniënt wanneer meerdere tijdsintervallen moeten worden vergeleken. Het triangulair model visualiseert de tijdsintervallen als punten in een tweedimensionale ruimte. Door deze visualisatie te gebruiken, kan een groot aantal tijdsintervallen vergeleken worden met één blik.

In deze thesis breiden we het triangulair model uit om onzekere tijdsintervallen voor te kunnen stellen en te kunnen behandelen. Een methode voor het verkrijgen van de relationele informatie tussen twee tijdsintervallen wordt tevens voorgesteld. Ten slotte werd een volledige implementatie ontwikkeld om te tonen dat de implementatie van het theoretisch model uitvoerbaar is. We voorzien een implementatie die imperfecte temporele informatie representeert, behandelt en opslaat. Ook de bevraging is geïmplementeerd door de implementatie van de temporele operatoren gedefinieerd in

het theoretisch model.

English summary

In information systems, the storage and handling of time-dependent data or data with a temporal component (for simplicity, we will refer to these as temporal data from now on) is of special interest. When dealing with temporal data in a database, it is necessary to modify the default behavior of the database engine. Usually, when dealing with temporal data, different versions of the same data are stored. In other words, the evolution of the data over time is stored. Therefore, a mechanism to store and handle the different versions of the stored data as well as to ensure the consistency among these versions should be provided.

In addition to this, the available temporal information is usually not perfect. Therefore, it is necessary to provide a formal tool to handle the imperfections in temporal information.

The studies of time in language and in knowledge lead us to the conclusion that human beings deal with time in an uncertain, imprecise and/or vague way. In this thesis, we propose a formal model to deal with imperfect time intervals, based on possibility theory and fuzzy set theory. Both theories provide very well-known formal tools to deal with uncertainty, imprecision and vagueness.

In this work, we study the treatment of imprecise temporal data in a database. To achieve this, we propose a theoretical model for fuzzy temporal relational databases. This model represents and handles imperfect temporal data in a consistent way. The main contribution of this model is an approach to the treatment of imperfect temporal information which lies closer to human reasoning. The proposed model is complete. We provide the necessary data types, integrity constraints, data definition language (DDL) and data manipulation language (DML). The model solves the main issues of dealing with temporal information in a database. It is possible to store several versions for the same data. The consistency mechanism is provided through the data manipulation language (DML), which is redefined to ensure the consistency of the temporal data. By doing this, we ensure database consistency, even in the presence of imperfect temporal information.

In this thesis, we study the flexible querying of temporal data. First, comparison operators are extended through the use of specific temporal operators (before, after, during, ...). Usually, a Boolean value is obtained as the result of an evaluation. However, in the possibilistic framework, possibility and necessity degrees in the unit interval [0, 1] are obtained as the result of an evaluation. These degrees provide the user with more information than a Boolean value.

In order to provide for a more powerful tool to represent user preferences, the bipolar querying of databases is proposed. There are two main approaches in literature. Following the first one (called the constraint-wish approach), a user specifies a set of constraints which must hold for the selected objects. The user may also provide a set of properties which are also desirable for the selected objects. For example: a user may want a car which should be dark and is wished to be black. Following the second approach (called the satisfaction-dissatisfaction approach), a user may specify a set

of constraints which must hold for the selected objects and another set of constraints which must not hold for the selected objects. For example: a user may want a car which should be dark but should definitely not be blue.

In this thesis, we use the satisfaction-dissatisfaction approach to query temporal databases in a bipolar way. This has appeared to be useful in both historical and criminal databases.

When a query is being processed in application to a database, the processing system has to make some calculations in order to determine whether an object fulfills the query constraints or not. The main problem here is the aggregation of the evaluation of non-temporal constraints and the evaluation of temporal constraints. Moreover, in the case of bipolar querying, we either have to aggregate the evaluations of both constraints and wishes or the evaluations or both satisfaction and dissatisfaction. One of the main issues when showing query results to a user is their ranking. The results which are more interesting to the user should be given a higher score than the results which are less interesting to the user. Therefore, in this work, we provide several methods for the aggregation of the evaluations of temporal and non-temporal constraints. Next to that, we provide some methods to rank query results.

Usually, a big amount of data is obtained as the result of a query. Nowadays, the most modern database management systems provide tools to summarize this information and to present query results in a user-friendly way. When querying temporal data, showing the validity period of the queried data is of special importance. Usually, this temporal information is visualized as an interval corresponding to some time line. However, this representation is not convenient when multiple time intervals have to be compared. The triangular model visualizes time intervals as points in a two-dimensional space. Using this visualization, a big number of time intervals can be compared in a single glance.

In this thesis, we extend the triangular model to be able to represent and handle uncertain time intervals. A method to obtain the relational information between two time intervals is also proposed.

Finally, in order to show that the implementation of this theoretical model is feasible, a complete implementation has been developed. We provide an implementation which represents, handles and stores imperfect temporal information. Querying is also implemented through the implementation of the temporal operators defined in the theoretical model.

Resumen en Español
"Tratamiento de la información difusa temporal en bases de datos relacionales"

En los sistemas de información actuales, es de gran interés el almacenamiento y la manipulación de datos que tengan una componente temporal o bien sean dependientes del tiempo (por simplicidad, a partir de ahora, nos referiremos a ellos genéricamente como *datos temporales*). Para modelar esta información en una base de datos, hay que tener en cuenta que la naturaleza temporal de los datos implica una serie de alteraciones en el comportamiento natural de los mismos, ya que nos vemos obligados a mantener varias versiones de un mismo objeto, o lo que es lo mismo, su evolución a lo largo del tiempo. Por lo tanto es también necesario proporcionar mecanismos para almacenar, tratar y asegurar la consistencia entre las distintas versiones de los datos almacenados.

Otro problema adicional es que la información temporal de la que se dispone, no siempre es conocida con precisión. Por lo tanto, a los mecanismos mencionados anteriormente, hay que dotarlos de herramientas para manejar esta particularidad.

En esta tesis, estudiamos primero cómo entendemos y tratamos el tiempo los seres humanos. La conclusión a la que llegamos es que, los humanos somos capaces de hacer razonamientos muy precisos con expresiones temporales vagas e imperfectas. Esto nos conduce a presentar un modelo matemático para modelar y razonar con intervalos de tiempo, basado en la teoría de la posibilidad y la lógica difusa, herramientas fundamentales que permiten modelar la imprecisión y la vaguedad.

En este trabajo, estudiamos el problema del tratamiento de la información temporal imprecisa en una base de datos. Para ello, proponemos un modelo teórico para bases de datos relacionales, siguiendo la aproximación posibilística. Este modelo permite almacenar y tratar datos temporales de una manera consistente. La principal novedad con respecto a otros modelos de bases de datos temporales es el manejo que hacemos del tiempo, lo que permite representar y razonar con datos temporales de una manera más cercana a como lo hacemos los humanos.

El modelo que proponemos es completo ya que definimos las estructuras de datos, restricciones de integridad y proporcionamos los lenguajes tanto de definición de datos (DDL) como de manipulación de datos (DML). Nuestra propuesta resuelve también los problemas clásicos que encontramos en bases de datos temporales. Primero, permitimos que existan varias versiones para los datos temporales y aseguramos su consistencia redefiniendo las operaciones del lenguaje de manipulación de datos (DML). De esta manera, y a pesar de que las expresiones relativas al período de validez de los datos se proporcionen de manera imprecisa, el contenido de la base de datos es consistente.

Por otra parte, estudiamos también la consulta flexible de los datos temporales almacenados en la base de datos. Primero extendemos los operadores clásicos de comparación con operadores temporales del tipo *anterior, posterior, durante, ...,* etc. En un

enfoque clásico, los operadores de mencionados devuelven *verdadero* o *falso*, pero al utilizar el marco teórico posibilistico, podemos emplear varias medidas para el cómputo del grado de cumplimiento de una comparación. Este grado, a diferencia de los operadores clásicos, nos proporciona un grado de cumplimiento de la condición, que normalmente se suele encontrar en el intervalo $[0, 1]$. Esto nos proporciona mayor información como resultado de nuestras consultas a la base de datos.

Con la intención de captar de una manera más completa las preferencias de los usuarios, se propone la consulta bipolar de bases de datos, que permite dar un mayor poder expresivo a los usuarios. Existen dos propuestas que, como veremos, son equivalentes. En la primera (llamada restricción-deseo), el usuario expresa por un lado un conjunto de restricciones que son de obligado cumplimiento, y por otro lado un conjunto de condiciones que serían deseables. Por ejemplo, podemos buscar un coche de color oscuro, preferiblemente negro.

En la otra propuesta de consulta bipolar (llamada satisfacción-insatisfacción), el usuario especifica por un lado un conjunto de restricciones que satisfacen la consulta y por otro lado, un conjunto de restricciones que no son deseables. Por ejemplo, podemos buscar un coche de color oscuro, pero que no sea azul.

En esta tesis, extendemos la segunda propuesta para consultas bipolares en bases de datos temporales. Como veremos es útil en aplicaciones de bases de datos históricas y también en búsqueda criminalística.

Cuando se ejecuta una consulta en una base de datos, el sistema ha de realizar una serie de cálculos para determinar si un determinado objeto satisface o no los criterios de la consulta. En este caso, nos encontramos con el problema que supone la agregación de la evaluación de los criterios temporales y la evaluación de los criterios no temporales. Además, en el caso de la consulta bipolar, tenemos el problema de la agregación de las restricciones y los deseos (si utilizamos la primera propuesta) o la agregación de la satisfacción e insatisfacción (si utilizamos la segunda propuesta).

A la hora de mostrar los resultados a los usuarios, nos encontramos con el problema de ordenar los resultados, de modo que los objetos más interesantes sean los primeros que se le muestren al usuario. Por lo tanto, en este trabajo proponemos varios métodos para agregar primero los resultados procedentes de criterios temporales y los resultados procedentes de criterios no temporales. Finalmente proponemos mecanismos para ordenar los resultados y presentarlos al usuario.

Suele ocurrir que, como resultado de una consulta, obtenemos un conjunto de datos muy grande que hay que mostrar al usuario. Hoy en día, los principales sistemas de bases de datos, cuentan con herramientas para resumir y presentar de manera más amigable dicha información. Sin embargo, cuando tratamos con datos temporales, como parte de los resultados de una consulta, hay que mostrar el período de validez de dichos datos. Normalmente, ese intervalo de tiempo se visualiza como una línea cuya longitud es proporcional a la duración del intervalo. Cuando queremos comparar un gran número de intervalos esta visualización no es viable. El modelo triangular propone una visualización de los intervalos de tiempo en dos dimensiones. Esto hace que un intervalo de tiempo en el modelo triangular, se represente como un punto. La ventaja que proporciona es que podemos estudiar las relaciones entre un gran número de intervalos de un simple vistazo.

En este trabajo, extendemos el modelo triangular para representar intervalos de tiempo con incertidumbre. Presentamos también una manera visual de obtener las relaciones entre intervalos de tiempo que presentan imprecisión.

Por último, con la intención de demostrar que el modelo teórico de bases de datos desarrollado en esta tesis es factible en un sistema real, hemos realizado su imple-

mentación completa, extendiendo para ello un sistema de bases de datos relacional. Dicha implementación permite tanto representar datos temporales con imprecisión así como realizar consultas utilizando los operadores de comparación estudiados para datos temporales.

1 Introduction

Contents

The temporal dimension is a constant in our daily life. Practically, every aspect of our daily life is joined by the temporal dimension. But, although the concept of time is well known, it is very complex to define it.

More formally, it could be said that the time is a physical magnitude. There is no agreement on the starting of the time (which is related to the Big Bang) neither on the ending of the time (although some scientists believe that the time could be infinite). The perception of the time is associated with the culture. In some cultures, there is a linear perception of the time whereas in some other cultures there is a cyclical perception. Because of this, the time is usually organized in hierarchical and cyclic structures, for example the calendars. In a calendar, the time is organized in one or several hierarchical levels. Each one of these levels are called by some authors, granules. For example, the Gregorian Calendar [1] organizes a year in twelve months.

Despite of these hierarchical and well organized temporal structures, human beings handle the time in an imprecise way. Very often, while communicating, some vague expressions are used and refer to a specific temporal moment. For example, "We will see each other tomorrow around ten ". This last sentence, contains imprecision; the exact time for the meeting would be approximately between five to ten minutes before or after ten o'clock. On the other hand, the sentence is also an example of ambiguity. It is not explicitly set whether the meeting would be in the morning or at night. Nevertheless, both speaker and hearer, would not have any doubts in the time for the meeting.

The linguistic aspect of the time [2, 3] has been studied in depth as well as the semantics of the time in language [4–8]. The main motivation for these studies is to build a model that handles temporal aspects. In 1983, Allen [9] studied all the possible relationships between two time intervals.

As a result of the study of time in language, several conclusions have been achieved. First of all, in a temporal indication, two different types of time are distinguished. Speaker Time, ST is the time when the temporal indication was uttered. Next to that, referred time, RT is the time mentioned in the temporal indication. It has been proven that the higher distance between both ST and RT, the bigger the rounding that humans beings do. For example, consider the sentence "I lost my home keys last week". The time interval associated with the time indication "last week" is approximately around 5 to 9 days. Nevertheless, in the sentence "Romanticism lasted around one hundred years", the time interval is somewhere around eighty and one hundred and twenty years. RT and ST are illustrated in Figure 1.1.

Another main source of imprecision in language is the granularity; The changes between different granularities lead to a precision loss. For example consider the sentence "I was born the thirteenth of September of 1983". The temporal indication in the sentence has a granularity of days. If we change the granularity to months, then the sentence would be "I was born on September 1983". By doing this, we are introducing some imprecision in the sentence, since any day of September is possible for the temporal indication. Figure 1.2 illustrates the example.

The term **information** comes from the Latin word *informatio* which means to give form, to form an idea. But, as explained in [10], information is a polymorphic phenomenon and a polysemantic concept. The reader is refered to [11] for an easy-to-read introduction.

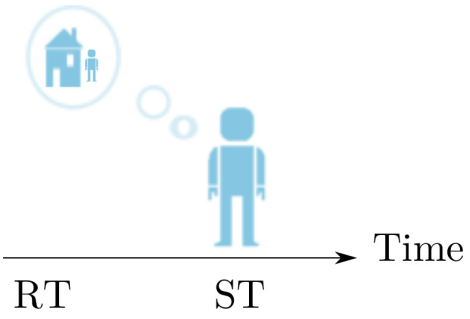


Figure 1.1: Speaker Time vs Refered Time

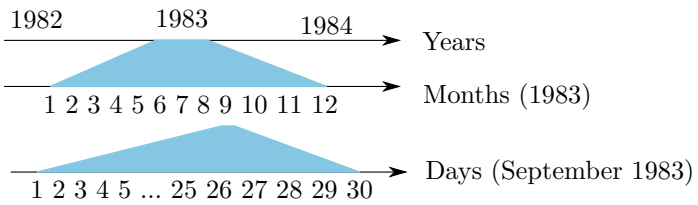


Figure 1.2: Example of imprecision in granularity shifts.

An Information System (*IS*) is defined as a combination of technology and people's activities [12], [13]. That technology supports, handles and manages the information. In other words, *IS* includes software, hardware, data, people and procedures.

Usually, at the backbone of each Information System is a Database Management System (*DBMS*) The main features of any database system are twofolded, and they are intrinsically connected. The first one is the storage of data. The data are organized within a model. Several models have been proposed. Among them, the most popular is the (object) relational model [14, 15]. The second one is the querying. The users may query the data stored in the database. In order to achieve that, a query language is provided. The most popular query language is the Structured Query Language, *SQL* [16].

As it can be seen, both storage and querying are interconnected. Whereas the storage needs a representation model, the querying needs a language to make queries.

As a natural evolution of the relational model, several proposals has been done in order to represent more complex types of information. The other way around, in order to implement more realistic models, some extensions of the relational model are proposed. Two main extensions were, first, the representation of incomplete / missing information. Second, the representation of temporal information associated to the data. At this point, it is possible to distinguish between the period of time in which the data is valid, e.g., the time associated to the validity of a physical constant, like the temperature of a room, and some other time associated to the data. E.g., the time when a bank transaction is made.

To allow information systems to cope with these and similar data imperfections, many approaches adopt fuzzy sets [17] for the representation of temporal information [18–21]. The temporal relationships studied by Allen were fuzzified by several authors [19, 22, 23]. Garrido et al. [24] present different temporal operators, defined

by a combination of regular fuzzy comparisons. Also, [24, 25] studied uncertainty in temporal expressions concerning time intervals. Other approaches, like [26], use rough sets [27] to represent imperfect time intervals.

The concept of time itself is very complex to handle and interpret [2, 3], though it is very natural and omnipresent. As information systems attempt modelling natural objects, concepts or processes, they often require modelling temporal aspects or concepts. Thus, several proposals have arose to obtain theoretical models that allow the modelling or representation of time [7, 8].

In reality, some aspects or properties of objects or concepts are time-variant or time-related, e.g., the moment of a bank transaction is traditionally a point in time and thus a time-related notion, the function of an employee in a company can change through recorded history and is thus time-variant. A temporal database schema is a database schema that models real objects or concepts with time-related or time-variant properties. However, the modelling of temporal aspects has a direct impact on the consistency of the temporal database, because the temporal nature of these aspects imposes extra integrity constraints. An example. Consider a library database, modelling the stock of books in the library. Two dates are stored: the loan and the return date. It is clear that a book cannot be loaned again until it is returned. Without further precautions, a library employee could loan the same book several times even if it is not returned. A temporal database model will typically constrain record insertion and prevent similar modelling inconsistencies.

Figure 1.3 shows a schematic diagram with some of the main database types in information systems which are relevant for this work. This schema does not show all the different database systems. For example, spatial databases that offer support for Geographical Information Systems GIS are not shown.

The representation of incomplete information in the relational model [28] has been studied in depth [29–33]. Some of these proposals deal with the representation and the semantics of null values in the relational model.

The modelling of imperfect information has been done by using one of the three main theoretical frameworks. First, probability theory [34, 35], possibility theory [17, 36–39] and rough sets [27].

Possibility theory [17] has been the key in the development of fuzzy databases. There are several proposals [40–48] to represent and handle imperfect or vague information in a database. Flexible querying has been also studied in depth [49–51]. Recently, some extensions to query with both positive and negative criteria have been proposed [52]. This is also known as bipolarity.

Query languages have been proposed to query the database in a flexible way. For example, *SQLf* [53] and *FSQL* [54]. Usually an implementation of these languages is provided. For example, *Freedom-0* [55], *SQLf* [56] and *FSQL* [54]. For further details we recommend the book on fuzzy databases [57].

A lot of research concerns temporal database models and their approaches to the modelling of time. The first efforts were towards the representation of historical information related to objects represented by records in a database [58]. Some proposals tried to extend the Entity Relationship Model [59], without impact on any database standards like SQL [60].

There are some theoretical models for probabilistic databases which extend the

relational model [61]. There are also some proposals for a probabilistic algebra [62–64]. These models are implementated in real systems like MayBMS [65–69]. Spatio-temporal databases represent temporal information [70–72] as well as geographical information.

Notably, in 1994, "A Consensus Glossary of Temporal Database Concepts" was published [73]. For this publication, 44 temporal database researchers, among which some of the main researchers in this field, cooperated to reach a consensus on the nature and definitions of some of the main temporal database concepts and terminology. This glossary was subsequently updated in 1998 [74].

An interesting issue in temporal modelling concerns relationships between temporal notions. In this sense, Allen [9] studied temporal relationships between time intervals (and as a special case time points). Among others, the querying of temporal databases has greatly benefited from these temporal relationships, because they support richer and more complex user-specified temporal query demands, by allowing to express more complex relationships between the temporal notions in the temporal expressions in the query. For example, a query like 'who were the department heads when Thomas worked for the institution' can be evaluated using operators similar to Allen's ones.

Some authors proposed the modelling of time intervals by using a fuzzy set [18, 19, 21, 75]. In 2009, Garrido et al, [24] proposed a compact representation for time intervals in a fuzzy database. Some operators to compare temporal intervals were also provided. At the same time, Qiang [26] proposed a technique to model imperfect time intervals by using rough sets [27].

Over the last few years, a model to deal with imperfect information has come back. Uncertain databases deal with uncertainty and / or possibilistic data. The possible worlds model [76] had as a main drawback the computational complexity with respect to the query evaluation. Nevertheless, twenty years later, Bosc et al. [77–79] extended the model so the queries are now evaluated in a compact and more efficient way.

Figure 1.4 shows the positioning of this work with respect to the other approaches.

1.1. Objectives

The main objectives of this thesis are the following.

1. Definition and formalization of temporal data types and operators. The goal is to abstract the main characteristics of the temporal data types. Then, it would be possible to re-define the data models and the operations in a database.

   As part of this main objective we find the following objectives.

   a) To define a representation for the temporal elements. The chosen representation should allow uncertainty and vagueness within the temporal elements.

   b) To define the operations and semantics for the possible relationships between the temporal elements defined above.

   c) To extend the fuzzy relational model to represent the temporal elements and to support the operations on them.

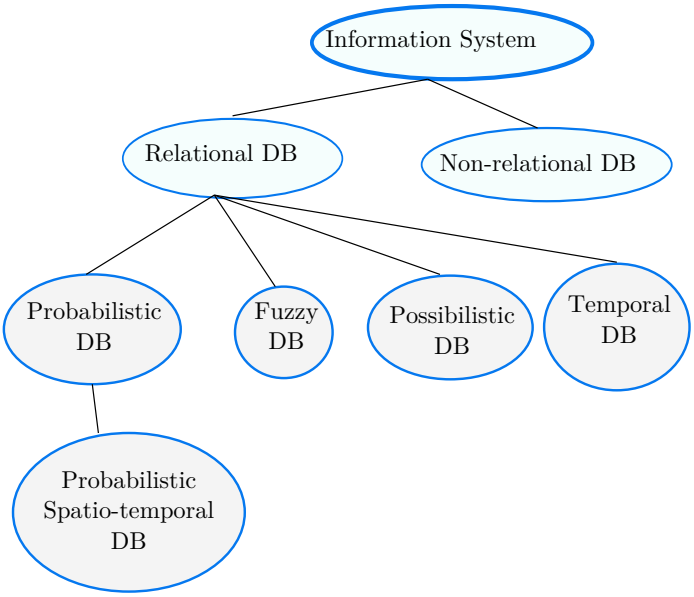


Figure 1.3: Diagram with the main database types in information systems.

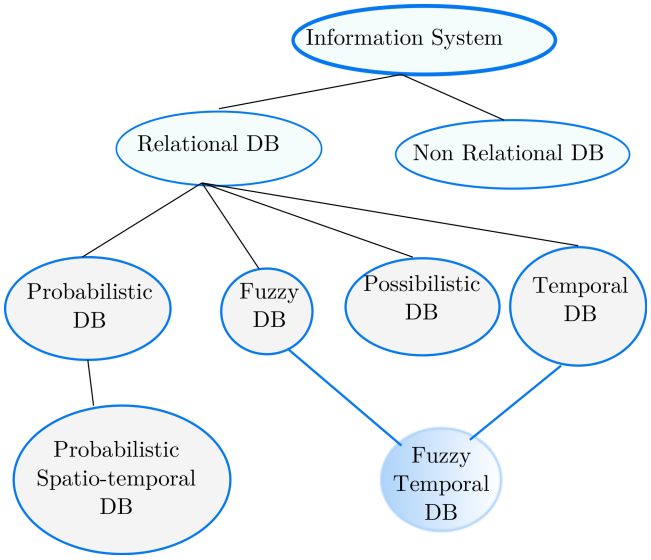


Figure 1.4: Our proposal with respect to the present approaches.

d) To define the behaviour and the semantics for the time-dependent data.

2. Implementation of the theoretical model obtained in the previous stage.

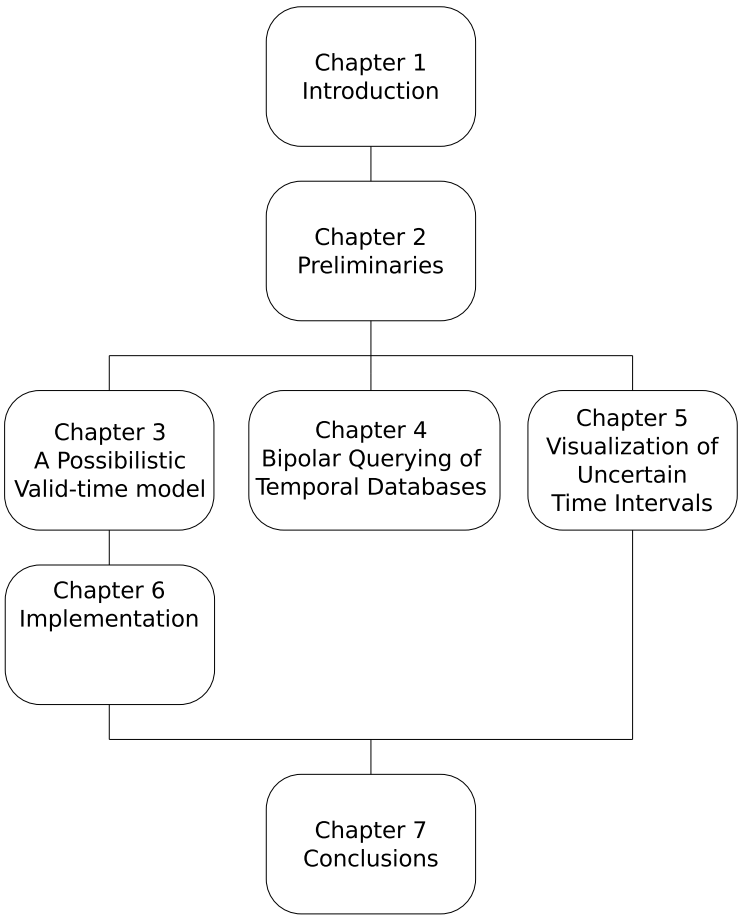


Figure 1.5: Contents of each chapter and relationships between them.

1.2. Contents

Figure 1.5 illustrates the organization of the chapters in this thesis. Most of the content has been already published. The references here refer to the section "List of Publications". The contents are organized as follows:

Chapter 1: The problem is introduced as well as the objectives and the outline of the work.

Chapter 2: The previous works and the state of the art are introduced. First of all, we will explain some basics aspects of time modelling and the imperfections associated with time modelling. Then, the behaviour and some of the main

temporal database models. Next to that, we will introduce the main theoretical framework: the fuzzy logic. This will serve us to explain and introduce the fuzzy databases. Some basic concepts about bipolarity will be introduced at the end of the chapter. The contents of this chapter are partially covered by a book chapter [16], and a conference proceedings [9].

Chapter 3: In this chapter, the model to represent imperfect temporal data is defined. We will extend the relational model by redefining the behaviour in the temporal operations. Finally, the model will be implemented in the top of a fuzzy databases model called GEFRED. The contents of this chapter are partially covered by two journal papers [1] and [2] and several conference proceedings [6, 7, 14].

Chapter 4: In this chapter, we present bipolarity as a querying tool. The state of the art is studied with two main models: the constraint-wish model and the satisfaction-dissatisfaction model. Finally we will propose some extensions to the satisfaction-dissatisfaction model to query temporal databases. A practical example is given by using the presented approach to query historical databases. The contents of this chapter are partially covered by a book chapter [17] and several conference proceedings [10, 4, 15].

Chapter 5: In this chapter, we present the triangular model to visualize time intervals. We propose an extension to represent uncertain time intervals and to study the relationships between them. The model is compared with the theoretical model presented in Chapter [3]. The contents of this chapter are partially covered by several conference proceedings [3] and [8].

Chapter 6: This chapter is about the implementation of GEFRED in an open source framework for object-relational mapping, Hibernate. Both representation and querying capabilities are implemented within the framework. The contents of this chapter are partially covered by a book chapter [18] and several conference proceedings [11, 12, 13].

Chapter 7: In this chapter we will conclude with the main contributions of the work. We will finish the work with the further research work.

1.3. A comment on the notation

This thesis is the result of the collaboration between two research groups. In this work, we have tried to find a balance between two goals. First, to keep the notation as in the original publication. Second, to provide a consistent notation through the whole thesis. If both objectives are in a conflict, then we try to provide a consistent notation.

There are several cases where the notation and / or the name of certain elements have to be clarified. In particular, in Chapter 3, the notation and the naming of some elements for the relational database model could be confused with some existing literature. To clarify this, we will describe the different namings that have been proposed in the literature.

The implementation of some operators use the notation from the relational model. Often, a mathematical set is usually notated as a capital letter, for example S. The elements of the set are then notated with a lower case letter. Therefore, a set and its elements are given by $S = \{s_1, \ldots, s_n\}$. Nevertheless, because we follow the relational notation, a set is notated with a lower case letter in Chapter 3.

In order to provide a consistent notation through the thesis, in Chapter 4, we have modified the notation and the naming from the original publications. In this case, we follow the relational notation and the naming from the previous chapter.

1.4. List of Publications

1.4.1. Journal Papers

1. J. E. Pons, N. Marín, O. Pons Capote, C. Billiet, and G. De Tré, "A relational model for the possibilistic valid-time approach," *International Journal of Computational Intelligence Systems*, vol. 5, no. 6, pp. 1068–1088, 2012.

2. A. Bronselaer, J. E. Pons, G. De Tré, and O. Pons, "Possibilistic evaluation of sets," *Int. J. Uncertainty Fuzziness Knowlege-Based Syst.*, vol. 21, no. 3, 2013.

1.4.2. International Conferences

3. C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, "A comparison of approaches to Model Uncertainty in Time Intervals" in *Proceedings of the EUSFLAT conference*, Sep. 2013

4. C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, "Bipolar querying of Valid-time intervals subject to Uncertainty" in *Proceedings of the Flexible Querying Answering Systems conference*, Sep. 2013

5. C. Barranco, J.M. Medina, J. E. Pons and O. Pons Capote "Building a Fuzzy Valid Time Support Module on a Fuzzy Object-Relational Database" in *Proceedings of the Flexible Querying Answering Systems conference*, Sep. 2013

6. J. E. Pons, C. Billiet, O. Pons Capote, and G. Tré, "A possibilistic valid-time model," in *Advances on Computational Intelligence* (S. Greco, B. Bouchon-Meunier, G. Coletti, M. Fedrizzi, B. Matarazzo, and R. Yager, eds.), vol. 297 of *Communications in Computer and Information Science*, pp. 420–429, Springer Berlin Heidelberg, 2012.

7. C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, "Evaluating possibilistic valid-time queries," in *Advances on Computational Intelligence* (S. Greco, B. Bouchon-Meunier, G. Coletti, M. Fedrizzi, B. Matarazzo, and R. Yager, eds.), vol. 297 of *Communications in Computer and Information Science*, pp. 410–419, Springer Berlin Heidelberg, 2012.

8. G. de Tré, A. Bronselaer, C. Billiet, Y. Qiang, N. van de weghe, P. de Maeyer, J. E. Pons, and O. Pons, "Visualising and handling uncertain time intervals in a

two-dimensional triangular space," in *Proceedings of the 2nd World Conference on Soft Computing*, 12 2012.

9. J. E. Pons and O. Pons, "Uses of fuzzy temporal databases on information systems," in *Proceedings of the IADIS International Conference Informations Systems* (P. P. Miguel Baptista Nunes, Pedro Isaías, ed.), pp. 429–432, March 2012.

10. C. Billiet, J. E. Pons, T. Matthé, G. De Tré, and O. Pons Capote, "Bipolar fuzzy querying of temporal databases," in *Lecture Notes in Artificial Intelligence*, vol. 7022, (Ghent, Belgium), pp. 60–71, Springer, Octobre 2011.

11. J. E. Pons, I. Blanco Medina, and O. Pons Capote, "Generalised fuzzy types and querying: implementation within the hibernate framework," in *Proceedings of the 9th international conference on Flexible Query Answering Systems*, FQAS'11, (Berlin, Heidelberg), pp. 162–173, Springer-Verlag, 2011.

12. J. E. Pons, O. Pons Capote, and I. Blanco Medina, "A fuzzy valid-time model for relational databases within the hibernate framework," in *Proceedings of the 9th international conference on Flexible Query Answering Systems*, FQAS'11, (Berlin, Heidelberg), pp. 424–435, Springer-Verlag, 2011.

13. J. E. Pons, O. Pons, and I. Blanco Medina, "An open source framework for fuzzy representation and querying in fuzzy databases," in *Proceedings of the IADIS International Conference Informations Systems* (P. P. Miguel Baptista Nunes, Pedro Isaías, ed.), March 2011.

1.4.3. National Conferences

14. J. Pons, A. Bronselaer, O. Pons, and G. de Tre, "Possibilistic evaluation of fuzzy temporal intervals," in *Actas del XVI congreso Español sobre Tecnologías y Lógica Fuzzy* (Valladolid, Spain), february 2012.

15. J. E. Pons, C. Billiet, O. Pons, G. de Tré, E. de Paermentier, and J. Deploige, "Consultas bipolares en bases de datos temporales: aplicación en bases de datos con datos históricos," in *Actas de las Jornadas Andaluzas de Informática 2011.*, pp. 150–156, 9 2011.

1.4.4. Book Chapters

16. J. E. Pons, C. Billiet, O. Pons, and G. de Tré, *Dedicated to Patrick Bosc*, ch. Aspects of Dealing with Imperfect Data in Temporal Databases.

17. C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, *Eleventh International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets*, ch. Bipolarity in the Querying of Temporal Databases.

18. J. E. Pons, O. Pons, and I. B. Medina, *New trends on intelligent systems and soft computing*, ch. Fuzzy temporal information treatment in relational DBMS. Theoretical Formulation, Implementation and Applications, pp. 95–112.

1.4.5. Manuscript under review

19. J. E. Pons, J.M. Medina, C. Barranco and O. Pons Capote “A Fuzzy Temporal Model built on a Fuzzy Object-Relational Database” *Expert Systems with Applications*

2

Time modelling, temporal databases and fuzzy databases

The contents of this chapter have been partially published on:

- J. E. Pons, C. Billiet, O. Pons, and G. de Tré, *Flexible Approaches in Data, Information and Knowledge Management* , ch. 9 Aspects of Dealing with Imperfect Data in Temporal Databases. 2013.
- J. E. Pons and O. Pons, "Uses of fuzzy temporal databases on information systems," in *Proceedings of the IADIS International Conference Informations Systems* (P. P. Miguel Baptista Nunes, Pedro Isaías, ed.), pp. 429–432, March 2012.

Contents

In this chapter, we will introduce some basic concepts and the state of the art in both time modelling and temporal databases.

We will start introducing the basic concepts and properties of time modelling in information systems. Temporal granularities and temporal relationships are also studied in detail. Next to that, the most common types of imperfections in time are analysed. Then, we study the representation of imperfect temporal information with special attention to the imperfections in temporal relationships.

Once the time modelling is explained, we will provide a detailed overview of the main proposals, classifications and difficulties in the temporal database research. This chapter finishes with an overview of the most popular commercial temporal database systems.

2.1. Time Modelling

Before considering the introduction of temporal modelling to information systems, it is necessary to define and explain some main concepts concerning temporal modelling, and to discuss some properties and issues related to these concepts. Most of these concepts are widely used in the community of temporal databases and their definitions have been agreed upon in the context of [73]. For these concepts, in the entire chapter, the proposals of [73] are followed (and often cited).

2.1.1. Basic Concepts and Properties

In information systems, time itself is usually perceived as a linear or cyclic concept [80]. Therefore, a time domain is usually represented by a set with an imposed partial order. In general, two main types of time models can be discerned: a *linear* model [81] and a *cyclic* model [82]. In the linear model, a total order is imposed on the set and the progress of time is seen as a linear matter, while cyclic models are mainly used in the modelling of recurrent processes. It should be noted that the majority of proposals use linear time models.

Data models used by information systems (more specifically, temporal database systems) may represent these underlying time domains using *chronons* [73], which can informally be described as the smallest distinguishable time units that can be used in the system. However, to explain what chronons are, an explanation of some other temporal concepts is necessary.

Definition 1. Instant [73]
An ***instant*** is a time point on an underlying time axis.

Thus, an instant is basically an instantaneous time point in a time model. Then, it is possible to define the concept time domain.

Definition 2. Time domain [83]
A ***time domain*** is an ordered pair $(\mathcal{T}, \leq)$ where $\mathcal{T}$ is a non-empty set of time instants and $\leq$ is a total order on $\mathcal{T}$. A time domain is said to be discrete if all the elements other than the last have an instant successor, and all the elements other than the first have an instant predecessor.

Orthogonal to the classification of underlying time models as linear or cyclic, they can be classified as *discrete*, *dense* or *continuous* models [73]. In a discrete model [58], the notion exists that every instant has a unique successor and the set of instants is seen as a discrete one. Here, intuitively, the set of instants can be seen as isomorphic to the set of natural numbers $\mathbb{N}$. In a dense model, the notion exists that between any two instants always lies another. Here, intuitively, the set of instants can be seen as isomorphic to the set of rational numbers $\mathbb{Q}$ (when the set of instants is a discrete one) or the set of real numbers $\mathbb{R}$ (when the set of instants is a continuous one). In a continuous model, the notion also exists that between any two instants always lies another one, but the set of instants is always seen as continuous and there are no "gaps" between successive instants.

Some other relevant concepts are:

Definition 3. Time Interval [73]
A ***time interval*** is the time between two instants.

Definition 4. Duration [73]
A ***duration*** is an amount of time with known length, but no specific starting or ending instants.

A time interval as such is bounded by two instants, whereas a duration is not. Also, it should be noted that an instant is in fact a singular case of a time interval where the start and end point are the same.

Definition 5. Temporal Element [73]
A ***temporal element*** is a finite union of time intervals.

Definition 6. Event [73]
An ***event*** is an instantaneous fact, i.e. something occurring at an instant.

Data models for time modelling might represent an underlying time domain using chronons:

Definition 7. Chronon [73]
In a data model, a ***chronon*** is a non-decomposable time interval of some fixed, minimal duration.

A data model may now represent a time model or time domain by a sequence of consecutive chronons. These chronons have identical durations. A data model will typically not specify the exact chronon duration, so it can be fixed later by the applications.

The fact that chronons are actually time intervals has a particular effect on the representation of instants and time intervals. In a data model representing a time domain using chronons, an instant is of course represented by a chronon. A time interval may be represented by a set of contiguous chronons, depending on the amount of time the time interval comprises.

Another classification of time models concerns the use of points or intervals to model time. The equivalence between interval-based and point-based time models is demonstrated in [84].

Restrictions on time range may exist, as time may be bounded orthogonally in the past and in the future [80].

2.1.2. Granularities

An important issue in time modelling concerns the concept of *temporal granularities*. Informally, we may say that a granularity is the precision on the time expression. For example, consider a speaker saying *"My birthday is in May"*, the speaker works with a granularity of months. However, it is possible for the speaker to say more exactly *"My birthday is the 15th of May"*. A formal definition for this concept is given in [85]:

Definition 8. Granularity [86]
A ***granularity*** is an ordered set of non-overlapping and continuous temporal elements called *granules*. It can be said that a granularity is a partition in the set of chronons.

Definition 9. Granule [86]
A ***granule*** is the basic time unit in a granularity.

A temporal granularity is in fact a partitioning of the time line (time model) used by a system, usually dependent on the application. For example, the age of an adult human being is usually expressed in years: one will use sentences like 'Laura is 21 years old' instead of sentences like 'Laura is 21 years, 3 months and 4 days old'. In this example any duration shorter than a year needs no specific representation and thus the used granularity allows no specification for durations shorter than a year. The granules are years.

As a granularity G is an ordered set, each granule may be represented by an integer. In this representation, to keep track of the granularity a granule is an element of a granularity G. The corresponding granularity name is added in subscript:

$$G = \{i_G \mid i \in \mathbb{Z}\} \tag{2.1}$$

In a system, the granularity with the shortest granules is the *chronons granularity*, which is denoted by '$\perp$'.

Definition 10. Mapping function for granularities [85]
A mapping function f is a function that maps a given granule i_G, $i \in \mathbb{Z}$, in a given granularity G, to a set of corresponding chronons:

$$\begin{aligned} f &: G \to \mathcal{P}(\perp) \\ i_G &\mapsto \{c_\perp \mid (c_\perp \text{ is contained in } i_G) \wedge (c_\perp \in \perp)\} \end{aligned}$$

Where $c_\perp$ is a granule in the granularity of chronos, that is, $c_\perp$ is also a chronon.

Note that a mapping function f always maps from a given granularity G to the powerset $\mathcal{P}(\perp)$ of the set of chronons $\perp$. Therefore, the output for a mapping function is an element of $\mathcal{P}(\perp)$ and thus a subset of $\perp$.
A mapping function f requires that the following properties hold [85]:

- *G is an ordered set.*
- *G is a set of continuous granules.*

- *The granules in G do not overlap.*

The existence of mapping functions between granularities and the chronons granularity also allows comparing granularities with respect to the length of their granules. In this context, two important concepts can be discerned.

Definition 11. Finer Than [85]
Consider a mapping function f and let i_G and j_H be elements of granularities G and H respectively. Granularity G is now said to be ***finerthan*** granularity H if:

$$\forall i_G \in G, \forall j_H \in H : |f(i_G)| < |f(j_H)|$$

Definition 12. Coarser Than [85]
Consider a mapping function f and let i_G and j_H be elements of granularities G and H respectively. Granularity G is now said to be ***coarser than*** granularity H if:

$$\forall i_G \in G, \forall j_H \in H : |f(i_G)| > |f(j_H)|$$

It is also possible to describe the relation between different granularities. This is called a casting function, which is defined below:

Definition 13. Casting function [85]
Consider two different granularities G and H. A granularity-to-granularity ***casting function*** $cast$ is then a function mapping granules from G to granules from H:

$$cast : G \times \mathbb{G} \times \mathbb{G} \rightarrow H$$
$$(i_G, G, H) \rightarrow j_H$$

where $i_G \in G$ and $j_H \in H$ and where $\mathbb{G}$ denotes the set of all granularities.

Thus, the function $cast$ associates a granule i_G in G to a corresponding granule j_H in H. Two kinds of granularity-to-granularity mappings can now be discerned: an *upwards mapping* is a mapping from a granularity G to a coarser granularity H, whereas a downwards mapping is a mapping from a granularity K to a finergranularity L. Besides this classification, mappings between two granularities may be classified as *regular mappings*, *irregular mappings* or *congruent mappings* [85], [87].

- *Regular mapping*: A regular mapping is a granularity-to-granularity mapping where the mapping function value is calculated by means of multiplications and/or divisions and (maybe) an anchor adjustment, e.g., the mapping value of the mapping between hours and minutes is calculated using a multiplication by 60.
- *Irregular mapping*: An irregular mapping is a granularity-to-granularity mapping where the mapping function value can not be calculated by means of multiplications and/or divisions, e.g., the mapping value of the mapping between months and days is dependent on the exact month or day.
- *Congruent mapping*: A congruent mapping is a granularity-to-granularity mapping where the two granularities involved in the mapping have the same granules but a different anchor, e.g., the mapping between the days (Gregorian calendar days) and the academic days is a congruent mapping.

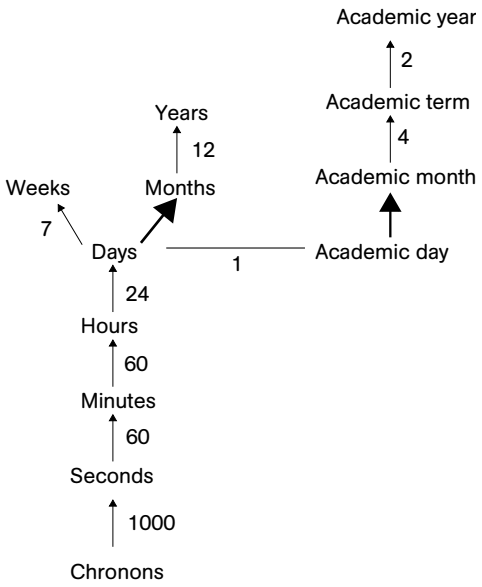


Figure 2.1: The granularity graph corresponding to Example 1.

Different granularity-to-granularity mappings between several granularities can be represented using a granularity graph, which is a directed graph indicating the mapping conversions. The above is illustrated in the following example.

Example 1. Consider a system that models both Gregorian calendar dates as well as academic calendar dates. In this system, the chronons granularity is a set of milliseconds. Figure 2.1 shows the complete granularity graph corresponding to this example. The transition between the chronons granularity and the seconds granularity is an example of a regular mapping. Regular mappings are represented by thin arrows in the visualisation of the graph. The transition between the days granularity and the months granularity is an example of an irregular mapping. In the graph visualisation, irregular mappings are represented by a bold arrow. Finally, the transition between the Gregorian calendar day granularity and the academic day granularity is an example of a congruent mapping. Both concern the same days, but the academic year starts on October 1st, whereas the Gregorian calendar year starts on January 1st. In the graph visualisation, congruent mappings are visualised as straight lines without arrow heads.

Next, in subsection 2.1.3, some basic concepts and issues concerning calendars are explained.

2.1.3. Calendars

A calendar provides some sort of human interpretation of time. As such, calendars provide meaning or interpretation to temporal values where this meaning or interpretation is relevant or useful to the user. More precisely, calendars determine the mapping between human-meaningful time values and an underlying time model [73]. Indeed, a calendar is an organization of different granularities (e.g., granularities day, month and

year) in a hierarchy. In order to formally define what a calendar is, a linear hierarchy of Granularities should first be introduced [88]. In order to ensure the compatibility with the modelling of granularities, some of the definitions in [88] are adapted to follow the formalization of the granularities introduced in the previous section.

Definition 14. Linear Hierarchy of Granularities [88]
A ***linear hierarchy of granularities*** is a finite set of granularities with a linear order (denoted $\sqsubset$). The most coarse granularity is called *the top of the hierarchy* and must be an infinite set while all other granularities must be finite sets.

An example of a linear hierarchy is the set $\{$days, months, years$\}$, where days $\sqsubset$ months and months $\sqsubset$ years. With this, a *linear calendar* can be defined.

Definition 15. Linear Calendar [88]
A ***linear calendar*** consist of a linear hierarchy of granularities and a validity predicate which specifies if a granule is valid in the calendar.

An example of a linear calendar could be the linear hierarchy $\{$ days, months, years $\}$, where days $\sqsubset$ months and months $\sqsubset$ years, together with a validity predicate that excludes time indications like '32 April 2012' or '30 February 2012'.

The linear hierarchy days $\sqsubset$ months $\sqsubset$ years represents a linear calendar. It is necessary to define the validity predicates because 17 January 2012 is a valid element of the calendar whereas 30 February 2012 is not.

In order to represent a Gregorian Calendar, a linear calendar is the most appropiate structure. Nevertheless, there are some other types of calendars, like periodic calendars. For further reading on this topic, we refer the reader to [88].

2.1.4. An example of temporal domain: Julian Day Number

The Julian Day number *JDN* [89] is a counter. Its value is incremented in one unit every day from 1 January 4713 B.C. at 12:00 noon. The particularity of starting at noon was useful for astronomers: the observations they took one night belonged to the same Julian Day.
Note that the Julian Day represents whole days. There is an extension that allows to represent any precision needed (it is called Julian Date). By default, a Julian Day is expressed in Universal Time. (U.T., also known as Solar Time). However, there are also representations in Terrestrial Time (T.T.) and Epheris Time (J.E.D. or J.D.E.). Any time scale different from Universal Time must be explicit after the Julian Day number. There are several conversion formulas [90] [91] [92] between a date in Gregorian calendar format and a date in JDN format. The inverse conversion formula is proposed in [93].

There are many alternatives to optimize the representation of Julian Day numbers because of its extremely far origin (4713 B.C. year). Table 2.1 shows several time domains that can be calculated from the Julian Date, some of them are proposed just for optimization purposes.

Name	From	Formula	Current Value[1]
Julian Date (JD)[a]	12:00 noon Monday 1 January 4713 B.C		2455278. 85488
Julian Day Number (JDN)[b]	12:00 noon Monday 1 January 4713 B.C.	JND = floor(JD)	2455278
Reduced Julian Day (RJD)[c]	12:00 noon Tuesday 16 November 1858 A.C.	RJD = JD - 2400000	55278.85488
Modified Julian Day (MJD)[d]	00:00 Wednesday 17 November 1858 A.C.	MJD = JD - 2400000,5	55278.35488
Truncated Julian Day (TJD)[e]	00:00 Friday 24 May 1968 A.C.	TJD = JD - 2440000,5	15278.35488
Truncated Julian Day (TJD)[f]	00:00 Thursday 10 November 1995 A.C.	TJD = (JD- 0,5) mod 10000	5278. 35488
Dublin Julian Day (DJD))[g]	12:00 noon Sunday 31 December 1899 A.C.	DJD = JD - 2415020	40258. 85488
Chronological Julian Day (CJD)[h]	00:00 Monday 1 January 4713 B.C.	CJD = JD + 0,5 + timezone adjust.	2455279. 3548843 (UT)
Lilian Day Number[i]	Friday 15 October 1582	floor(JD-2299160,5)	156118
ANSI Date[j]	Monday 1 January 1601	floor(JD-2305812,5)	149466
Rata die[k]	Monday 1 January 1 A.C	floor(JD - 1721424.5)	733854
Unix time[l]	Thursday 1 January 1970 A.C.	(JD – 2440587.5) $\times 86400$	1269333062

[1] Current value on 23/03/2010.
[a] This is an extension of the Julian Day that allows time representation.
[b] Each day changes at noon.
[c] Used by astronomers.
[d] It starts at midnight.
[e] Definition from NASA. [94].
[f] Definition from NIST. [95].
[g] Introduced by the IAU in 1995.
[h] The timezone must be explicited. Each day changes at midnight.
[i] The number of days since Gregorian calendar in Universal Time.
[j] The origin for COBOL integer dates.
[k] The number of days since actual era.
[l] It counts the seconds not the day.

Table 2.1: Julian Day representations

2.1.5. Temporal Relationships

In this section, a brief introduction concerning *temporal relationships* can be found, sometimes also called 'temporal relations' [20]. Temporal relationships can be seen as relationships between temporal elements belonging to the same time domain. These relationships express how the temporal elements are related to one another, with respect to temporal precedence and overlap.

First the relative positions between a time point and a time interval were studied. There are three main relationships: the time point is before the time interval, the time point is during the time interval and finally, the time point is after the time interval. If we extend this study to the case of two time intervals, then, the relation during can be subdivided into three relations: overlaps, during and meets.

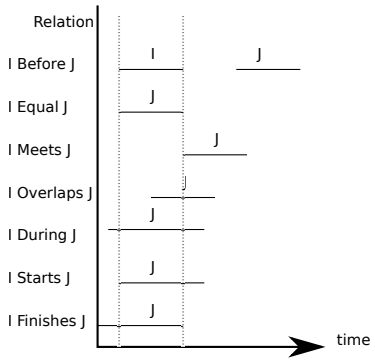


Figure 2.2: Allen relations between two crisp time intervals I and J. Note that the crisp time interval I is fixed and the different positions of the interval J illustrate the Allen relations.

Allen [9] most notably described such relationships between time intervals and as a special case, between instants. Figure 2.2 shows the seven basic temporal relationships that Allen discerned. Table 2.2 shows the implementation of the thirdteen Allen relations between two time intervals I and J.

As explained in the introduction, humans handle temporal information using temporal notions like time intervals or time points [73]. While the used temporal notions may contain imperfections [5], [21], [19], [96], humans often gracefully deal with these, as their inherent interpretation capability accounts for a lot of them. This phenomenon has been studied a.o. in the field of artificial intelligence [97], [98] and language understanding [6], [19], [5]. An information system, however, cannot appeal to a similar interpretation functionality. Thus, many proposals have been concerned with the combination of time and imperfections in the context of information systems [19]. Some proposals can be applied to both crisp and other time intervals [22], [19], [23], [24]. In this section, some main concepts and issues concerning this combination are presented.

Name	Implementation
I equals J	if $s_i = s_j \wedge e_i = e_j$
I starts J	if $s_i = s_j \wedge e_i < e_j$
I started by J	if $s_i = s_j \wedge e_i > e_j$
I finishes J	if $s_i > s_j \wedge e_i = e_j$
I finished by J	if $s_i < s_j \wedge e_i = e_j$
I meets J	if $e_i = s_j$
I met by J	if $s_i = e_j$
I overlaps J	if $s_i < s_j \wedge e_i < e_j \wedge e_i > s_j$
I overlapped by J	if $s_i > s_j \wedge e_j < e_i \wedge s_i < e_j$
I during J	if $s_i > s_j \wedge e_i < e_j$
I contains J	if $s_i < s_j \wedge e_i > e_j$
I before J	if $e_i < s_j$
I after J	if $s_i > e_j$

Table 2.2: Allen relations represented in the framework. $I = [s_i, e_i]$, $J = [s_j, e_j]$

2.1.6. Types of Imperfections in Temporal Modelling

As studied by Motro [99, 100] there are different sources of imperfection in an Information System.

- *Error*: Erroneous information is the simplest form of imperfection in information. An information system contains erroneous information when the stored information is different from the true information. One specific type of erroneous information is *inconsistency* which happens when the same aspect of the modeled reality is represented more than once.

- *Imprecision*: The imprecision in an information system happens when the information stored represents a set of possible values and the real value is one of the elements of the set. Hence, imprecision in information is different from erroneous information. There can be distinguished several types of imprecision like disjunctive information (e.g., Tom's birthday is either 10th or the 11th of July) , range information (e.g.,Tom's birthday is between June and July), negative information (e.g., Tom's birthday is not on August), and information with error ranges (e.g., Tom's birthday is the 11th of July $\pm$ 1 day).

- *Uncertainty*: It is the case when the knowledge about the information can not be stated with absolute confidence (e.g., Tom's birthday is probably the 11th of July). In this case, the confidence about the information should be qualified with a measure of confidence.

Taking this classification into temporal modelling, a distinction is made between the following sources of imperfections [19].

- *Uncertainty*: Temporal information or data may contain uncertainty. This means that the exact temporal value is (partially) unknown, however, generally some knowledge is present anyhow, possibly describing the value [21], [19], [96], [18]. E.g., the temporal notion described in a sentence like 'The Belfry of Bruges

was finished on a day somewhere between 01/01/1201 A.D. and 31/12/1300 A.D.' contains uncertainty: it is known that the belfry of Bruges was finished on a single day and that this day lays somewhere between 01/01/1201 A.D. and 31/12/1300 A.D., but it is not known exactly which day it is.

- *Vagueness*: Temporal information or data might contain inherent vagueness, as a precise instant or time interval may be intended, but the description of it is certainly vague [23], [19], [5]. E.g., the temporal notion described in a sentence like 'It happened during summer.' contains vagueness, as even the boundaries of the mentioned temporal notion are not clearly expressed.

- *Subjectivity or ambiguity*: Temporal notions might be subject to subjectivity or ambiguity. In certain cases, the temporal notion concerns a historical period like 'late romanticism' or 'the early middle ages' and thus contains subjectivity [19]. In other cases, the interpretation of the temporal notion depends on extra factors. E.g., consider a person saying to another person 'Let's meet each other at six.' The person hearing these words doesn't now if 6 a.m. or 6 p.m. is intended, though the person saying the words does.

As to the sources of the imperfections in temporal information, most proposals consider no specific source [98], [23], [19], [96], [18], [22], [101]. Some proposals, however, deal with the imperfections specifically resulting from aspects of language [5] and other proposals consider transitions between different granularities to be the source of imperfection in temporal information [85]. Therefore, some proposals consider granularity as the base of the temporal model [8].

In an information system, temporal information is usually related to facts or events [102]. In light of this, several classifications of temporal information can be considered, in which the following types of temporal information may be found.

A first classification refers to the time handled by a temporal expression:

1. Time position. E.g., yesterday.

2. Frequency. E.g., every Monday, two times a week.

3. Duration. E.g., 8 hours, the whole day.

Apart from this classification, a time indication can be classified in the following:

1. Relational time indications refer to a relation with respect to a time point or interval. For example after three hours, around May.

2. Situational time indications point to a time fact itself. For example in June, at 12 o'clock.

An orthogonal classification could be done in terms of the bounds of the temporal expression:

1. Bounded temporal expressions refer to the past, present or future. For example: yesterday at ten PM., in June 2013.

2. Unbounded temporal expressions do not refer to past, present or future. For example: at three o'clock.

Finally, an additional classification of temporal information is given by [64, 103]:

1. *Definite temporal information*: Definite temporal information contains information describing a situation in which all time indications associated with some fact are absolute time indications. The temporal information is precisely known.

2. *Indefinite temporal information* [64]: Indefinite temporal information contains information describing a situation in which the time indication associated with some fact has not been fully defined. E.g., consider an event that in fact occurred but it is not known exactly when.

3. *Repetitive temporal information* [103]: Repetitive temporal uncertain information contains information describing a situation in which an infinite number of time indications are associated with some fact. This is usually found in recurrent events like meetings. E.g., consider meetings that take place every Wednesday at noon. Some systems (usually with different information providers) may dispute the occurrence and/or the duration of a fact.

These three types of temporal information may be affected by uncertainty.

2.1.7. Representation of Imperfect Information

As mentioned before, information systems may have to deal with time indications which contain imperfections. Even for some specific events or facts, the temporal indications may become imprecise. Therefore, a time point might be specified by means of a time interval of which the boundaries may not be precisely known. An example.

Example 2. Consider a speaker and a hearer. The speaker wants to make an appointment with the hearer. Now, consider the speaker saying:

'We will meet each other tomorrow around 10'

The hearer will now usually instinctively agree that the appointment will be in, e.g., the time interval between 9.55h and 10.05h.

The study of the semantics of 'around' in temporal [5] indications has shown that the size of the time interval associated with the imprecise specification of a time point depends on the distance with respect to the current time. E.g., consider now that the speaker is talking about something that happened *'during last week'*, then the hearer would consider a time interval of more or less 10 days.

Some proposals [104], [8], [19], [102] conclude that the best representation for incomplete temporal knowledge is therefore based on time intervals, even if they refer to a fact that happen at a time point. This means that, as Allen proposed in [9], the primitive units (the chronons) in a time domain, used in an information system should be intervals.

In order to represent and manage uncertain temporal information properly, several theoretical frameworks have been proposed:

- *Probability theory*: Probability theory [64], [67], [105] is usually employed when uncertainty concerning a time interval allows a probability distribution to be associated to the time interval. The use of probability theory is very usual in logistics information systems. E.g., *'The package will arrive at its destination on Monday morning with a probability of 0.8'*.

- *Possibility theory*: Using possibility theory [38], a possibility distribution is associated to the temporal fact or event. Possibility theory is widely used to model uncertainty and vagueness in time [21], [96], [4], [19]. Several works [23], [22] present fuzzy versions of the temporal relations proposed by Allen [9]. The aim of these works is generally to obtain a flexible way to compare uncertain, ill-known temporal intervals modeled by possibility distributions by means of temporal relationships. The study of imperfect temporal meta data is done in [106], [107]. In [108] a proposal to use temporal fuzzy linguistic terms in fuzzy databases is studied. Burney [109], [110] has studied recently the combination of fuzzy databases with temporal data.

- *Rough sets*: Rough set theory [27] has been used to represent uncertainty in time intervals. The two dimensional representation of time intervals and the temporal relationships between them has been studied in [26]. In [111] a rough set-based model for temporal databases is presented. The study of temporal relationships between rough time interval is studied in [112].

2.1.8. Imperfections in Temporal Relationships

As the existence of temporal relationships allows to compare temporal notions, many approaches have been concerned with finding similar temporal relationships, able to support imperfections in the temporal information. Imperfections can be described by temporal notions or even by the temporal relationships themselves [22], [19], [23], [21]. These approaches are often based on Allen operators [9]. Example 3 presents a short example concerning one of Allen relationships.

Example 3. Consider an event which takes place between time points A and B. Thus, the event comprises time interval $[A, B]$ (this is visualized in part (1) of figure 2.3). The classical Allen relationship 'after' will be true for all intervals starting in $]B, \infty[$ as shown in part (2) of figure 2.3. A fuzzified version of Allen 'after' operator is illustrated in part (3) of figure 2.3. The comparison between any two time intervals results in a possibility degree in the unit interval. The shape of the corresponding possibility distribution is shown in part (3) of figure 2.3.

Note that all the points strictly after the point B results in a possibility degree of 1 whereas there is an area near the point B in which the possibility degree runs smoothly between 0 and 1.

Consider now the interval given by $[C, D]$, illustrated in part (4) of figure 2.3. The user wants to know if $[C, D]$ is after $[A, B]$. The crisp version of the 'after' operator would return *'no'* as an answer. The fuzzy version for the same operator would return *'yes, with a possibility of 0.5'*.

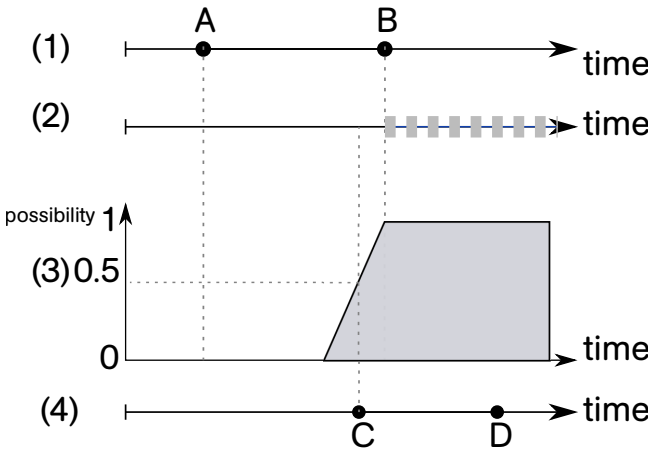


Figure 2.3: Example for the Allen relationship *'after'*. (1) The event bounded within time points A and B. (2) The crisp version of the *'after'* operator. (3) A fuzzy version of the after operator. (4) Another event, bounded within time points $[C, D]$.

2.2. Temporal Databases

A temporal database can generally be seen as a database that manages some temporal aspects in its schema [113], [20]. In subsection 2.2.1, some main concepts and properties concerning temporal databases and their definitions are presented and explained. In subsections 2.2.2 and 2.2.3, some issues of relational temporal databases are presented and discussed. Subsection 2.2.4 is an overview of the main temporal database models in the literature. Finally, subsection 2.2.6 presents an overview of some commercial temporal database systems.

2.2.1. Basic Concepts and Properties

A database schema models some part of reality. As mentioned in the introduction, the part of reality a temporal database schema tries to model, contains some temporal aspects. For example, in this part of reality, some concepts or objects could have time-related or time-variant properties. The modelling of these temporal aspects has to be handled specifically in order for the database to maintain a consistent model of reality.

Thus, a temporal database will contain *temporal values*, i.e. values representing (indications of) time. Temporal values in a temporal database can be classified into the following types based on their interpretation and modelling purpose. The definitions and explanations of these types can be found in [73] and [114] and more information can be found in [115], [116] and [114].

Definition 16. Valid Time [73]
The ***valid-time*** (VT) of a fact is the time when the fact is true in the modeled reality. For example, consider an application that stores the contracts for the employees in a company. The valid-time for each contract is a time interval with the starting and the ending dates of the contract.

Definition 17. Transaction Time [73]
A fact is stored in a database at some point in time, and after it is stored, it is current until logically deleted. The ***transaction-time*** (TT) of a database fact is the time when the fact is current in the database and may be retrieved. For example, consider an application that stores the contracts for the employees in a company. The transaction-time is a timestamp which stores the time when the data for the contract of an employee was stored in the database.

Definition 18. Decision Time [114]
Decision time (DT) denotes the time when an event was decided to happen. For example, consider the application that stored the contracts for the employees in a company. Decision time is the time when the decision to hire the employee was done.

Definition 19. User-defined Time [73]
User-defined time (UDT) is an uninterpreted attribute domain of date and time.

Valid times are usually provided by the user, whereas *transaction-times* are usually system-generated and -supplied [73]. Temporal values of type UDT are not given any extraordinary interpretation and have thus no extraordinary query language support [73].

A *temporal database* can now formally be defined as follows:

Definition 20. Temporal Database [73]
A ***temporal database*** supports some aspect of time, not counting user-defined time.

In a relational temporal database, temporal values will of course be in the tuples of the extensions, more specifically in temporal relations:

Definition 21. Valid-time Relation [73]
A ***valid-time relation*** is a relation with exactly one system supported valid-time.

Definition 22. Transaction-time Relation [73]
A ***transaction-time relation*** is a relation with exactly one system supported transaction-time.

A *valid-time*, respectively *transaction-time relational database* is now defined as containing one or more valid-time, respectively transaction-time relations [73]. Next to this, *bitemporal* relational databases contain both valid-time and transaction-time [73] and tritemporal or multitemporal databases contain valid-time, transaction-time and decision-time [114].

Figure 2.4 illustrates transaction-time with respect to valid-time. Valid-time is usually represented by a time interval whereas the transaction-time is represented by a timestamp. For a given tuple, there are the following options:

- The timestamp for the transaction-time is greater than or equal to the starting point of the interval for the valid-time, hence the tuple is said to be *immediate*. See Figure 2.5 (a).
- The timestamp for the transaction-time is after the valid-time interval, then the tuple is said to be *retroactive*. See Figure 2.5 (b).

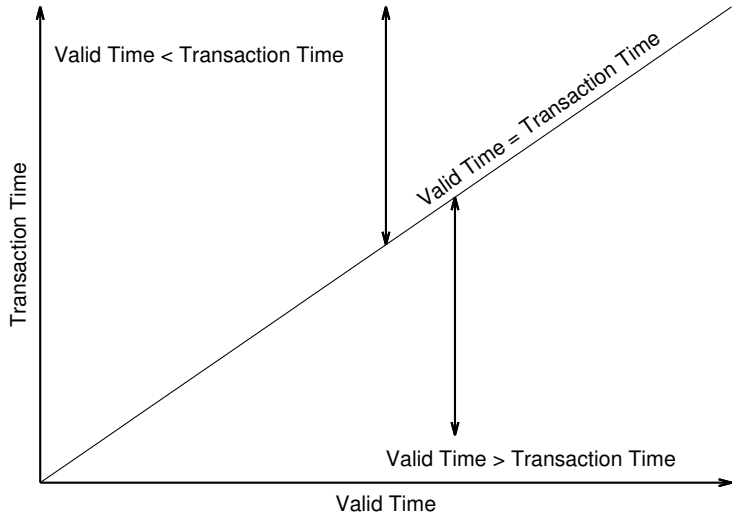


Figure 2.4: Valid-time with respect to Transaction Time.

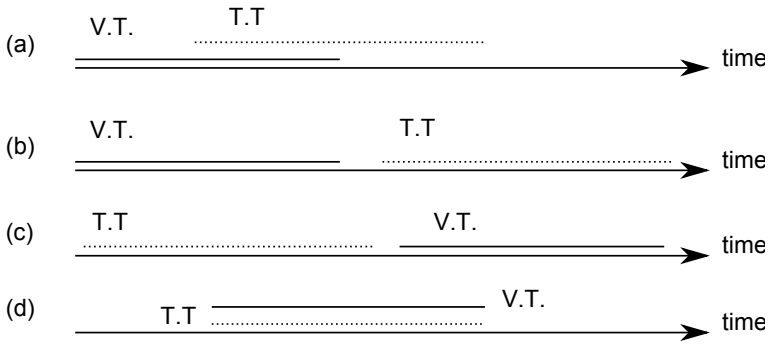


Figure 2.5: Illustration of the classification for valid and transaction-time. (a) Immediate: transaction-time is greater or equal to valid-time. (b) Retroactive: the transaction-time is after the valid-time interval. (c) Proactive: the transaction-time is before the valid-time interval. (d) Retro-proactive: valid-time is equal to transaction-time.

- The timestamp for the transaction-time is before the valid-time interval, then the tuple is said to be *proactive*. See Figure 2.5 (c).
- In the case that the valid-time interval crosses the line in which valid-time is equal to transaction-time, the tuple is said to be *retroactive* and *proactive* at the same time. See Figure 2.5 (d).

Figure 2.6 shows a 3D representation in which valid, transaction and decision-time are represented in each axis. Note the plane that divides the time space. The only feasible times are below that plane, this is because of the constraint that decision-time is always before transaction-time. In other words, it is not possible to store in the database (transaction-time) a fact before the decision to make that fact is done (decision-time).

We will illustrate the uses for each type of time with an example.

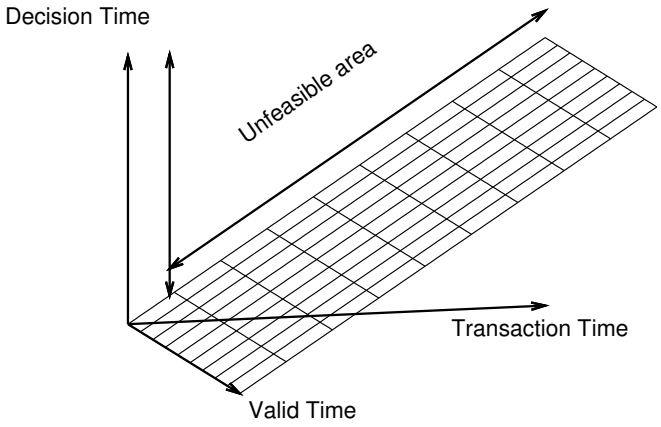


Figure 2.6: 3D representation for valid, decision and decision-time. The upper values from the shape are not feasible since decision-time is always before or during transaction-time.

Example 4. Consider an employee database containing data regarding the history for the employees in a company as shown in Table 2.3. For each employee, the following data are stored: a unique identifier (ID), the name of the employee, the valid-time (VT) when the contract started and ended. The transaction-time (TT) when the data was stored in the database and finally, the decision-time (DT), which is the time when the decision for that contract was made.

ID	Name	Start	End	TT	DT
1	Peter	25/08/2010	30/08/2011	23/08/2010	22/08/2010
1	Peter	31/08/2011	27/09/2011	26/09/2011	15/05/2011
1	Peter	28/09/2011	-	25/09/2011	17/09/2011
2	Maria	03/04/1984	03/04/1990	03/04/1999	2/04/1984
3	John	21/02/1999	-	21/02/1999	21/02/1999
4	Sarah	29/11/1985	-	03/04/1999	05/07/1985

Table 2.3: Example relation modelling the employees of a company. Values for the Valid, Transaction and Decision time attributes are visualized here in 'dd/mm/yyyy' format.

Consider now the employee with ID = 1. The first contract started on 25/08/2010 and finished on 30/08/2011. The decision for hiring that employee was made on 22/08/2010 but the record was stored in the database on 23/08/2010. Then, the company wanted to renew the contract for this employee. The new contract starting and ending dates are 31/08/2011 until 27/09/2011. The decision was made several months before, on 15/05/2011, but the record was stored in the database on 26/09/2011 due to

the holidays. Once the present contract finished, the company wanted to hire again the employee with ID = 1. The new contract started on 28/09/2011 and has no ending date (this is shown in the table with a '-' symbol). The decision for this new contract was made on 17/09/2011 and the record was inserted on the database on 25/09/2011.

As we can see, the second contract was stored after the third contract. In this example, the decision-time helps to order the real order for the contracts.

A very extensive list of the most well-known temporal database models can be found in [117]. As it is of course necessary to define a consistent way to query the temporal data, there are several proposals concerned with query languages and query language adaptations for temporal databases like [118] and [119].

In the rest of the chapter, the focus will be on concepts and issues concerning valid-time relations and aspects of valid-time relations. For this reason, the next two sections will present and discuss some main issues concerning temporal databases, specifically applied to or presented in the context of valid-time relations.

2.2.2. Primary Keys in Valid-time Relation Design

Generally, when designing a relation based on a relational database model, a subset of the relation's attribute set is usually chosen as primary key. The values of a tuple for these attributes will then uniquely determine this tuple, hence no two distinct tuples may have the exact same values for every attribute in this primary key. Also, a primary key must keep the irreducibility constraint. This constraint ensures that there are no redundant attributes in the primary key.

Next to attributes unrelated to time, a valid-time relation schema will typically contain one or more attributes which model the valid-time aspects and behavior of the real objects and concepts modelled by the relation schema. In this work, these attributes are called *valid-time attributes*. In valid-time relation extensions, distinct tuples can exist containing exactly the same values for every attribute except the valid-time attributes. These distinct tuples represent distinct versions of the same real object or concept, valid during different time periods. To allow the existence of such tuples when designing a valid-time relation using a relational database model, the most common solution is to include the valid-time attributes in the primary key.

The following example illustrates this primary key issue.

Example 5. Consider the example valid-time relation visualized in table 2.4, which models when certain people worked as employees in a certain company and under whose supervision they worked during that time. The valid-time attributes 'Start' and 'End' describe the year when an employee started, respectively finished working for the company. For example, the last tuple visualized in table 2.4 represents that the employee represented by this tuple started working for the company in 2005 and finished in 2009. The attributes 'Name', 'Birthday' and 'Supervisor' describe respectively the name, birthday date and unique identifier of the supervisor of an employee in the time during which he or she worked for the company. When correct, the date of an employee's birthday never changes and as such, the modelling of birthday dates has no effect on the database consistency. The 'Birthday' attribute thus describes user-defined

ID	Name	Birthday	Supervisor	Start	End
1	Peter	24/10/1985	3	2010	2012
2	Maria	03/04/1984	3	2001	2008
3	John	21/02/1964	-	1999	2001
4	Sarah	29/11/1985	2	2005	2009

Table 2.4: Example relation modelling the employees of a company. Values for the 'Birthday' attribute are visualized here in 'dd/mm/yyyy' format.

ID	Name	Birthday	Supervisor	Start	End
1	Peter	24/10/1985	3	2010	2012
2	Maria	03/04/1984	3	2001	2008
3	John	21/02/1964	-	1999	2001
4	Sarah	29/11/1985	2	2005	2009
4	Sarah	29/11/1985	2	2010	2013

Table 2.5: Example relation after including the valid-time attributes in the primary key and adding a tuple.

time ,UDT values. The attribute 'ID' describes employee identifiers. For each tuple, this identifier (a number) uniquely identifies the employee represented by the tuple.

Now consider {ID} being the primary key and consider the company wanting to hire Sarah again in 2010. This would be represented by another tuple in the relation, containing value 4 for attribute 'ID'. The existence of such a tuple is of course not allowed by the primary key, because it would mean the existence of two distinct tuples containing value 4 for attribute 'ID'. This problem can now be solved by defining a new primary key: {ID, Start, End}, which allows for the existence of distinct tuples with value 4 for attribute 'ID',which is not allowed as long as they have different values for attributes 'Start' or 'End'. The resulting relation is shown in table 2.5.

2.2.3. Consistency under Modification

The solution presented in subsection 2.2.2 concerns relation design and consists of including the valid-time attributes in the primary key. Unfortunately, implementing this solution as such allows for the existence of records whose values imply inconsistencies with respect to the modelling of reality.

Consider a valid-time relation which the primary key can be partitioned into two sets of attributes. One set contains attributes totally unrelated to time, for which the values of a record allow to uniquely identify the object or concept represented by the record. The other set contains the valid-time attribute(s). Because the valid-time attribute(s) is(are) included in the primary key, the existence of distinct records with exactly the same values for all time-unrelated attributes and distinct values for at least one valid-time attribute is not prohibited. Thus, inserting such records into the relation is not prohibited either, even if the information represented by the values for the valid-time attributes shows clear inconsistencies. An example.

ID	Name	Birthday	Supervisor	Start	End
1	Peter	24/10/1985	3	2010	2012
2	Maria	03/04/1984	3	2001	2008
3	John	21/02/1964	-	1999	2001
4	Sarah	29/11/1985	2	2005	2009
4	Sarah	29/11/1985	3	2007	2008

Table 2.6: Example relation with records whose values for the valid-time attributes violate consistency.

Example 6. Consider the example valid-time relation visualized in table 2.6, in which is based on the relation visualized in table 2.4. The primary key is again {ID, Start, End}. The last record in the relation represents a person named 'Sarah' started working for the company in 2007 and finished in 2008, with supervisor 'John'. However, the fourth record represents the same person (the value for attribute 'ID' is the same) started working for the company in 2005 and finished in 2009, with supervisor 'Maria'. The intention is clear: Sarah worked in the company from 2005 to 2009, first for Maria, then for John, then again for Maria. It is of course possible for an employee to change supervisors, but it is impossible for a person to start working in the same company twice at different times, for different supervisors, without stopping to work for one in between, as it is impossible to stop working for a supervisor twice at different times, without working for another one in between. The valid-time information represented by the last record is clearly not consistent with the valid-time information represented by the fourth record, or vice versa.

The most usual approach to deal with this inconsistency problem is to adapt the DML used by the DBMS, as to enforce consistency towards time with respect to the modelled reality.

Example 7. Reconsider the problem presented in example 6. The inconsistency arises when the last record in table 2.6 is inserted. Because the record's values for the valid-time attributes differ from those of the fourth record, the last record is accepted. The DML statement used was (the table is called 'Employees'):

```
INSERT INTO Employees VALUES
 (4, 'Sarah', '29/11/1985', 3, 2007, 2008);
```

The inconsistency problem can now be solved by replacing this statement with:

```
UPDATE Employees SET 'End' = '2007' WHERE
 (ID = 4) AND (Start = 2005) AND (End = 2009);
INSERT INTO Employees VALUES
 (4, 'Sarah', '29/11/1985', 3, 2007, 2008);
INSERT INTO Employees VALUES
 (4, 'Sarah', '29/11/1985', 2, 2008, 2009);
```

The resulting relation is visualized in table 2.7.

ID	Name	Birthday	Supervisor	Start	End
1	Peter	24/10/1985	3	2010	-
2	Maria	03/04/1984	-	2001	-
3	John	21/02/1964	-	1999	2010
3	John	21/02/1964	-	2010	-
4	Sarah	29/11/1985	2	2005	2007
4	Sarah	29/11/1985	3	2007	2008
4	Sarah	29/11/1985	2	2008	2009

Table 2.7: Example relation updated maintaining consistency.

2.2.4. Temporal database models

There are several extensions to the relational model that include temporal elements. In the current section, we briefly describe these models and some of its most interesting features. Some models work only with valid or transaction-time, whereas a few models can handle both. The current study starts with the models supporting valid-time only, continues through the models which support transaction-time and finally deals with the bitemporal models.

2.2.4.1. Valid-time models

Most models support valid-time only.

Brooks [120]: Proposes a tridimensional view of a valid-time database.

Wiederhold [121] : Presented a time-oriented database, developed to work with medical applications. This model depicts relations as sets of entity-attribute-time-value quadruples. Time is associated with the visit number of the patient.

LEGOL 2.0 [122]: A language used in legislative writing. The temporal order of elements and the valid-times for objects are important. It was the first time-oriented algebra to be defined, and some of its features are found in subsequent algebras. Tuples in this model have two time attributes: Start and Stop. Each one indicates the start point and the end point of the interval where the tuple is valid.

Clifford-1 [123, 124]: Each relation schema has an additional time value called 'State'. Also, a Boolean attribute is added to indicate if this tuple exists for that state.

Ariav [125]: Temporally-oriented data model: a valid-time relation is a set of snapshot relation states indexed by valid-time. A calculus-based query language is associated with the model: TOSQL.

Navathe [118]: Proposes a temporal extension to SQL called TSQL, the temporal algebra associated to the temporal relational model. This model allows a relationship to handle time-varying attributes, and also relationships without time-varying attributes. The only restriction is that all attributes in the relation

must be of the same type. Objects are classified in snapshot relations and valid-time relations. In the latter, each tuple has associated to it an interval of validity with two points: time-start and time-end, like in the LEGOL 2.0 model.

Sadeghi [126]: Proposes a calculus-based valid-time language, HQL, where all objects represent valid-time relations. As in Navathe and LEGOL 2.0, two time points are required for the start and end of the interval. The model requires coalescing.

Sarda [60]: This model was designed to support the calculus-based model HSQL. The model associates valid-time with tuples. Objects can either be snapshot or valid-time relations. In this model, valid-time is considered to be closed on the left part of the interval and open on the right. Time is represented by an implicit attribute called 'Period' as a single non-atomic value.

The following time data models are not in first normal form (1NF), which means they might have multiple values per attribute. Even though these models are an extension of the 1NF, the representation is not in 1NF, but the operators work on valid-time relations, which are extensions of conventional relations.

Segev [127]: Defines the temporal data model, whose principal time structure is the time sequence. A time sequence is a surrogate value that identifies the object along with a sequence of time-value pairs. There are several types of time sequences depending on the semantics of the data represented.

Clifford-2 [58,128]: This model is a refinement of the previous one presented by the same author. The data model allows two types of objects: a set of chronons, termed a lifespan, and a valid-time relation in which each attribute and tuple is assigned a lifespan. A tuple is an ordered value containing the tuple value and its lifespan. Attributes are not atomic since an attribute value in a tuple is a partial function from the chronons' domain to the value domain of the attribute. Relations have key attributes and, at the same chronon, no two tuples are allowed to match the key attributes.

Tansel [58,129]: Designed to support the calculus-based query language (Hquel) and the time by example language. The model supports only one type of object: the valid-time relation. Four types of attributes are supported. If we take the time into account, attributes can be time-varying or non-time-varying. Then, attributes can be atomic-valued or set-valued. There is no need for attributes in the relation to be of the same type, and attribute values in a given tuple do not need to be homogeneous. A triplet containing an element from the attribute's value domain and the boundary points of the time interval represents the value of a time-varying atomic-valued attribute. A set-valued attribute is represented by a set of these triplets.

Gadia-1 [130, 131]: This homogeneous model allows two types of objects: valid-time elements and valid-time relations. The model requires that all attribute values in a given tuple be functions on the same valid-time element. Unlike intervals, valid-time elements are closed under union, difference, and com-

plementation. Bhargava presents an extension of this model to both valid and valid-time.

Gadia-2 [130, 132, 133]: Is an extension of the homogeneous model known as multihomogeneous model. Temporal elements may be multi-dimensional to model both valid and transaction-time. Attribute values are functions from temporal elements onto attribute value domains, but attribute values do not need to be on the same temporal element. The key attributes in the relations must be single-valued which respect to the interval of validity.

Lorentzos [134, 135]: The temporal relational model allows to specify different granularities and the support of periodic events. This model associates timestamps with individual attribute values rather than with tuples. Timestamps are explicit values updated directly by the user. The said timestamps represent either the chronon during which attribute(s) were valid or a boundary point of the interval of validity. In the model, several timestamps of different granularities can be used together in the specification of a chronon. This model has the particularity that some columns have a different meaning depending on the context.

2.2.4.2. Transaction-time models

The main feature of the transaction-time models is that they are append-only. Therefore, only new versions of the tuples are inserted into the database. In some database models, transaction-time is implicitly managed by the database, so the user does not have to deal with transaction-time.

Kimball [136]: This temporal model is called DATA. In it, the association between facts and times is implicit. Also, the update operations avoid the explicit use of time. However, transaction-time relations cannot be displayed but can only be depicted in the snapshots extracted from the transaction-time relations. At query language level, the association of facts and times is also implicit.

Stonebraker [137]: Proposes the Postgres temporal data model. Like in the previous model, the association between time and facts in the three model features (update language, query language and visualization) is implicit. In this model, the visualization is not restricted to snapshot relation states. Two timestamps are used when specifying the time for a given tuple that is current in the relation.

Jensen [115]: In the data model with time (DM/T) model, the association of facts and times is also implicit. DM/T contains a backlog for each user-defined transaction-time relation. This backlog contains the full history of the associated user-defined transaction-time relation. When an insert, delete or update operation is performed, the backlog for this relation is updated.

2.2.4.3. Bi-temporal, multitemporal datamodels

Bi-temporal models deals with both valid and transaction-time. Usually, the changes in the valid-time are recorded with a timestamp in the transaction-time. The main

model is TSQL2 [80]. Snodgrass [119] proposed the changes to support TSQL2 in the standard SQL3.

To the best of our knowledge, the only multitemporal model is given by Nascimento [114] where valid, transaction and decision-time are stored and handled.

2.2.5. Temporal Query Languages

The SQL query language provides a very limited support for temporal data. Hence, a considerable amount of research has been done to extend SQL with advanced temporal support. Despite of the efforts to extend SQL with temporal support, none of the proposals is part of the standard. In the following we will briefly describe the temporal query languages found in the literature.

IXSQL [138]: An extension of SQL2 for time intervals. Two operators are provided in order to map between time intervals and time points: Fold and Unfold operators. These operators are known to be computationally inefficient .

ATSQL [139, 140]: Is an extension of SQL with temporal statements. The model uses interval-timestamped values for the tuples and works with interval semantics. The most recent version, ATSQL2, is a temporally complete language because it extends the relational schema to store time-varying data.

TSQL2 [80, 141]: As explained before, TSQL2 is an extension of SQL2 that supports both time points and intervals. It is a complex model that deals with both valid and transaction-time.

TQuel [116]: This is a temporal extension for the Quel language. The only time supported is valid-time and it is represented by time intervals. The main drawback is that there is not an algebraic language defined.

HTQUEL [142]: This language is based on the TQUEL language. Time is represented as intervals. The model defines a complete and sound relational algebra.

SQL/TP [143]: This is a temporal extension to SQL which is point-based. Time is represented as time point timestamps.

TOSQL [125]: This is another temporal extension to SQL. The access to both previous and current data versions is allowed although the update sentence is not defined.

2.2.6. Commercial Temporal Database Systems

Several commercial temporal DBMS exist. Table 2.8 gives an overview of some of the more better known temporal DBMS and provides references for more information.

Oracle workspace manager [144] and TimeDB [145] are libraries for dealing with time in OracleDB. TimeDB and Postgree Temporal [146] are similar: both are simple implementations that implement a subset of the Allen operators and some operations for the creation and manipulation of temporal attributes (valid-time, transaction-time or

Name	Time Supported	Comments	Reference
Oracle Workspace Manager	VT, TT.	Package for Oracle DB.	[144]
TimeDB	VT, TT.	Interface for Oracle DB.	[145]
Postgree Temporal	VT.	Package for Postgree SQL.	[146]
Teradata	VT, TT.	Used for data-mining.	[147]
Secondo	VT, TT.	Spatio-temporal database.	[149]

Table 2.8: Commercial Temporal Database Systems.

both are supported). Teradata [147] is mainly a business intelligence system designed for data mining. Secondo [148] is an extensible database system in which the core of the database may be replaced by a customized algebra. It is designed for non-standard applications and it supports both valid and transaction-times.

The most complete implementation is Workspace Manager.

Unfortunately, none of these systems take data imperfections into account, neither in data storage, nor in querying.

2.3. Fuzzy Modelling

This section is devoted to introduce some basic concepts of fuzzy sets [36–38, 150, 151]. First we will introduce the concept of a fuzzy set. Finally we will introduce some operators defined on fuzzy sets.

2.3.1. Fuzzy sets

Fuzzy sets are a generalization of classical sets in which the membership of an element with respect to the set is not completely known. The generalization is done as follows.

1. The membership for an element with respect to a set is now a fuzzy concept. For some elements it is not clear whether they belong to the set or not.

2. The membership for an element with respect to a set is quantified by a degree which is usually known as membership degree for the element with respect to the set an it usually takes values in the unit interval $[0, 1]$.

Therefore, a mathematical set A on the universe of discourse Ω, is fully characterized by its membership function μ_A which associates the value 1 with each element of A and 0 with each other element of U.

$$\begin{aligned} &\mu_A : \Omega \rightarrow \{0,1\} \\ &x \mapsto 1 \leftrightarrow x \in A \\ &x \mapsto 0 \leftrightarrow x \notin A \end{aligned} \tag{2.2}$$

For the definition of a fuzzy set, the discrete set $\{0,1\}$ of the previous membership function is extended to the unit interval $[0,1]$. More formally, a fuzzy set is defined as follows.

Definition 23. Fuzzy set
A fuzzy set $\tilde{A}$ defined over a universe of discourse Ω is a set of pairs $\mu_A(x)/x$.

$$\tilde{A} = \{\mu_A(x)/x : x \in \Omega, \mu_A(x) \in [0,1]\} \tag{2.3}$$

where $\mu_A(x)$ is the membership degree of the element x with respect to the a set $\tilde{A}$. It is important to notice that

$\mu_A(x) = 0$ means that the element x does not belong to the fuzzy set $\tilde{A}$.

$\mu_A(x) = 1$ means that the element x fully belong to the fuzzy set $\tilde{A}$.

Usually, the function μ_A is given by a membership function. If the membership function provides values of the set $\{0,1\}$, then we have a crisp set.

Example 8. Let us consider the price in euros for a restaurant to be the universe of discourse for the concept "middle priced restaurant". The fuzzy set representing the concept "middle priced restaurant" could be given by the following membership function.

$$\text{middle priced restaurant } = \{1/10, 1/15, 0.9/20, 0.85/25, 0.75/30, 0.65/35\}$$

The label "middle priced restaurant" associated with the fuzzy set, is also known as linguistic label.

Definition 24. Linguistic label
A linguistic label is a label associated with a fuzzy set.

It is clear that the definition of the membership function for a given linguistic label could be subjective and it will also depend on the universe for the fuzzy set.

Depending on whether the universe of the discourse Ω is finite or not, the definition of the fuzzy set in terms of the universe is as follows.

- Ω is finite:

  $$\Omega = \{x_1, \ldots, x_n\}$$

  A fuzzy set $\tilde{A}$ is expressed in the following way.

  $$\tilde{A} = \{\mu_1/x_1, \ldots, \mu_n/x_n\} \tag{2.4}$$

  Where μ_i is the membership degree for the element x_i, with $i = 1, \ldots, n$. By default, the elements with membership 0 are not listed.

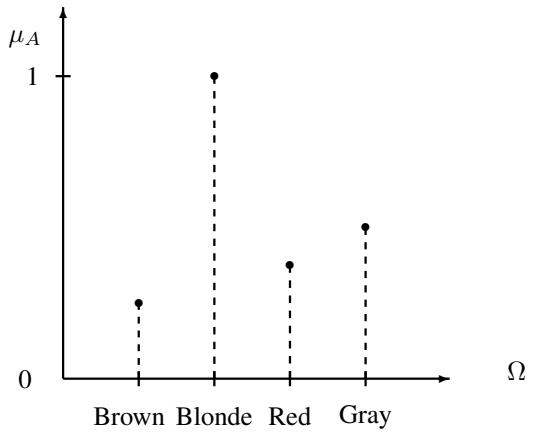


Figure 2.7: Discrete membership function for the "Blond hair" concept.

Example 9. Consider a set of hair colors {Black, Brown, Blond, Auburn, Chestnut, Red, Grey }. The concept " Blond hair" can be modelled by using a fuzzy set like $\tilde{A} = \{0.25/\text{Brown}, 1/\text{Blond}, 0.4/\text{Red}, 0.6/\text{Grey}\}$. The membership function is illustrated in Figure 2.7.

- Ω is infinite: The fuzzy set $\tilde{A}$ could be represented as in the following expression.

$$\tilde{A} = \int \mu_{\tilde{A}}(x)/x \tag{2.5}$$

where $\mu_{\tilde{A}}(x)$ is the membership degree of x.

Example 10. We want to model the concept "Tall" for the height of a person in meters. In this case, we could consider a continuous membership function since the universe of discourse Ω is also continuous. Figure 2.8 illustrates the membership function.

2.3.2. Concepts about fuzzy sets

The following operations and concepts are defined to handle fuzzy sets.

Definition 25. **α-cut (α-level set)**
The α-cut (α-level set) of a fuzzy set $\tilde{A}$ on a universe of discourse Ω is notated as $\tilde{A}_\alpha$ and defined for all the elements that fulfil the following condition.

$$\tilde{A}_\alpha = \{x : x \in \Omega, \mu_{\tilde{A}}(x) \geq \alpha\} \tag{2.6}$$

where $\alpha \in [0, 1]$.

The concept of strict α-cut (strict α-level set) is the following:

$$\tilde{A}_{\overline{\alpha}} = \{x : x \in \Omega, \mu_{\tilde{A}}(x) > \alpha\} \tag{2.7}$$

Two special cases of (strict) α-cut sets of a fuzzy set $\tilde{A}$ are the support (notated by supp($\tilde{A}$))and the core (notated by core($\tilde{A}$)).

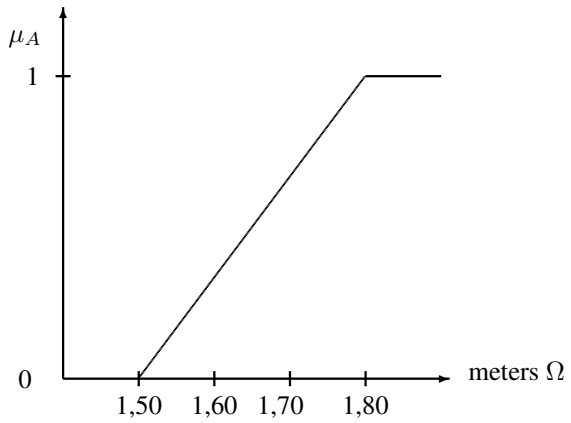


Figure 2.8: Continuous membership function for the "Tall" concept.

Definition 26. Support of a fuzzy set
The support of a fuzzy set $\tilde{A}$ on Ω is a regular set that fulfils the following.

$$\text{supp } \left(\tilde{A}\right) = \{x \in \Omega, \mu_{\tilde{A}}(x) > 0\} \tag{2.8}$$

Definition 27. Core of a fuzzy set
The core of a fuzzy set $\tilde{A}$ on Ω is a regular set that fulfils the following.

$$\text{core } \left(\tilde{A}\right) \{x \in \Omega, \mu_{\tilde{A}}(x) = 1\} \tag{2.9}$$

Definition 28. Height of a fuzzy set
The height of a fuzzy set $\tilde{A}$ on Ω is defined as follows.

$$\text{height } \left(\tilde{A}\right) = \sup_{x \in \Omega} \mu_{\tilde{A}}(x) \tag{2.10}$$

Definition 29. Normalized fuzzy set
A fuzzy set $\tilde{A}$ over Ω (see definition 23) is said to be normalized if:

$$\text{height } \left(\tilde{A}\right) = 1 \tag{2.11}$$

The cardinality of a fuzzy set is a real number which reflects a global membership degree of all the elements of the fuzzy set and therefore can not be considered as an indication of the number of elements of the fuzzy set. More formally, the cardinality of a fuzzy set is defined as follows:

Definition 30. Cardinality of a fuzzy set
Consider a universe of discourse Ω, if the universe is discrete, then the cardinality of the fuzzy set is computed as follows:

$$\text{card } \left(\tilde{A}\right) = \sum_{x \in \Omega} \mu_{\tilde{A}}(x) \tag{2.12}$$

If the universe of discourse Ω is continuous, then the cardinality of the fuzzy set is computed as follows:

$$\text{card}\left(\tilde{A}\right) = \int_{\Omega} \mu_{\tilde{A}}(x) \tag{2.13}$$

The standard complement $\overline{\tilde{A}}$ of a fuzzy set $\tilde{A}$ which is defined over a universe of discourse Ω is defined as follows.

$$\overline{\tilde{A}} = \{(x, 1 - \mu_{\tilde{A}}(x)) \,|\forall x \in \Omega : 1 - \mu_{\tilde{A}}(x) > 0\} \tag{2.14}$$

2.3.3. Interpretation of fuzzy sets

The membership degrees $\mu_{\tilde{A}}(x)$ of the elements $x \in \Omega$ of a fuzzy set $\tilde{A}$ can be interpreted in three different ways [152]:

1. Degree of compatibility. The fuzzy set has a conjunctive interpretation; the concept that is being modelled is represented by all the elements of the fuzzy set. Therefore, the membership function $\mu_{\tilde{A}}(x)$ returns a compatibility degree between the element x and the concept modelled by the fuzzy set $\tilde{A}$.

   For example, consider that we are looking for middle-priced restaurants. The membership function is defined in example 8. This set has a conjunctive interpretation: for us, all the elements of the fuzzy set are the concept "middle-priced restaurant".

2. Degree of truth / preference. In this case, the fuzzy set has also a conjunctive interpretation; all the elements of the fuzzy set represent the concept that is being modelled. The membership function $\mu_{\tilde{A}}(x)$ returns a degree that shows to what extent the element x of the fuzzy set $\tilde{A}$ applies. For example, consider the fuzzy set of the languages spoken by a person, where each membership degree represents a degree of truth. The fuzzy set can be specified by:

   $$\{ \text{(Spanish, 1), (English, 0.5), (Dutch, 0.2)} \}$$

   This membership function represents a perfect knowledge of Spanish, a medium knowledge of English and a basic knowledge of Dutch.

3. Degree of uncertainty. In this case, the interpretation of the fuzzy set is disjunctive. Usually this kind of fuzzy set models a variable taking exactly one single value. For some reason, that value is not exactly known and hence is modelled by a fuzzy set. The membership degree $\mu_{\tilde{A}}(x)$ represents the extent that the value x is the actual value of the variable modelled. For example, consider the average price p of a dish in a restaurant. If the price is not exactly known, but it is known that the restaurant is middle-priced then the interpretation of the membership degree is different than in the example related to the degree of compatibility. In this case, the interpretation is that the price p of the restaurant could be 20 euros to an extent of 0.9.

2.4. Fuzzy Databases

Humans manage knowledge in an imprecise way. It is a difficult task to represent knowledge when imprecision arises in Information Systems, IS [151, 153–155]. Several theoretical frameworks have been proposed to deal with this problem. Among them, probability theory [34, 35], possibility theory [17, 36–39] and rough sets [27]. Each one tries to represent and handle ill-defined information.

The main difference between possibility and probability theory is that the first one is less restrictive. Probability theory deals with incomplete knowledge due to *variability* in the outcomes of an experiment [38]. Possibility theory can capture incomplete knowledge whereas probability theory cannot [155, 156].

It is known that database management systems (DBMS) are the best way to store and retrieve data in an Information System. In order to store data, a representation model should be defined. Besides, in order to retrieve data, a query language is necessary. In the past decades, DBMSs have undergone a huge evolution. First of all, the representation of concepts (as entities) and relationships were first introduced by the relational model by [14], and [157]. Afterwards, several models to represent and query uncertainty were proposed.

First of all, an extension of the relational model was proposed by [15]. There are several proposals for fuzzy databases like [33, 40, 42, 44, 158]. Among them, the GEFRED model by [159] is a synthesis with the main features of the proposals mentioned before.

On the one hand, probability theory has also been used to represent uncertainty in information systems. The main theory was proposed in the work of [61] as an extension to the relational algebra. The management of probabilistic data was studied in [62]. An algebra for probabilistic databases is defined by [63] and [64]. In [70] a probabilistic temporal algebra is proposed. A more recent approach MayBMS, is proposed in [65]. This system is implemented on the top of PostgreSQL. As a consequence of the expansion of the WWW, several proposals deal with a XML database instead of a relational database. This has been studied by [66–68]. In [69] a mobile version based on XML is proposed.

Probabilistic spatio-temporal databases have been also studied by [70]. A logical formulation is proposed by [71] and a more recent approach by [72].

On the other hand, possibility theory has served as the basis for fuzzy databases. Abundant research on fuzzy relational databases has been done since the eighties by [40–47]. In [48] extended possibilistic truth values have been considered as an extension of the model.

The querying of a fuzzy database has also been studied in depth by [49–51], *SQLf* by [53] and more recently by [52] with the introduction of positive and negative criteria in the query specification.

There exist several implementations of these fuzzy databases systems like *Freedom-0* by [55], the query language *SQLf* has been defined in [56] and *FSQL* by [54, 160]. The FSQL server runs on top of Oracle DBMS, although there is also a version for PostgreSQL. A book for fuzzy databases that covers some aspects of the modelling, design and the implementation is [57].

Over the last few years, a new approach to incomplete information in databases,

called *Uncertain Databases* has been introduced. [161] makes a survey of current research trends in possibilistic logic. An interesting property is the way to handle both positive and negative information. Several theoretical models have been proposed. Among them, it is of special interest the possible worlds model first proposed by [76]. The main drawback for that model was the computational complexity of evaluating a query against all the possible worlds. The model has been extended twenty years later by [77–79]. In these works, the authors propose a new compact way to evaluate queries against all the possible worlds in an efficient way.

In the following subsection, we will briefly introduce and compare the main models and proposals for fuzzy databases in the literature.

2.4.1. Main models and proposals

The simplest approach consists on adding a membership degree for each tuple in a relation. That degree represents to which extent the tuple is in the relation. Usually, the degree is a value in the unit interval $[0, 1]$.

Example 11. Consider an employee database containing data regarding the history for the employees in a company, as shown in Table 2.9. For each employee, the following data are stored: a unique identifier (ID), the name of the employee and the starting and ending dates when the contract started and ended. A global membership degree is added to express the membership of each tuple to the relation. Therefore, the tuple representing the employee with ID = 1 and start and end dates 25/08/2010, 30/08/2011 respectively belongs to the relation with a degree of 0.8.

ID	Name	Start	End	μ
1	Peter	25/08/2010	30/08/2011	0.8
1	Peter	31/08/2011	27/09/2011	0.5
1	Peter	28/09/2011	-	1
2	Maria	03/04/1984	03/04/1990	0.23
3	John	21/02/1999	-	0.89
4	Sarah	29/11/1985	-	0

Table 2.9: Example of a simple fuzzy database. The relation models the data regarding the employees of a company. Values for the start and end time attributes are visualized here in 'dd/mm/yyyy' format. The membership degree for each tuple is given by μ.

In the previous example, it is not possible to distinguish cases where we are not sure about the name of the employee, or about the starting or ending date. For that reason, it would be more informative to know the membership degree of each attribute instead. The following models have been proposed in the literature:

2.4.1.1. Buckles & Petry

This model [41, 42] is based on the concept *similarity relation* as proposed by Zadeh [162]. A *Fuzzy Relationship* is defined as a subset of the following Cartesian product: $\mathcal{P}(D_1) \times \ldots \times \mathcal{P}(D_n)$, where $\mathcal{P}(D_i)$ is the power set of the domain D_i.

There are three main types of data within this model:

1. A discrete set of values. For example, consider the set {small, medium, big}.

2. A discrete set of numbers. For example, consider the set {20, 35, 40}.

3. A discrete set of fuzzy numbers. For example, consider the following labels associated with fuzzy numbers: { cheap, middle-price, expensive }.

The semantics for the data stored in the database are disjunctive. In other words, only one element in the given set is allowed for each value which are called *interpretations*. For each numeric or scalar domain D, a *similarity relation* is defined between each pair of elements in the domain. Usually, the values for a similarity relation are normalized within the interval $[0, 1]$ where 0 denotes *"completely different"* and 1 means *"equal or completely similar"*. Therefore, a similarity relationship can be defined as follows.

$$\begin{aligned} s_r : D \times D &\mapsto [0, 1] \\ s_r\left(d_i, d_j\right) &\mapsto [0, 1] \end{aligned} \tag{2.15}$$

with $d_i, d_j \in D$. A similarity relation is reflexive ($s_r(x, x) = 1$) and symmetric as in an equivalence relation. Special forms of transitivity prevail:

$$T1 : s_r(x, z) \geq \max_{y \in D} \{\min [s(x, y), s(y, z)]\} \tag{2.16}$$

$$T2 : s_r(x, z) \geq \max_{y \in D} \{s_r(x, y) * s_r(y, z)\} \tag{2.17}$$

Where $*$ is the arithmetic multiplication.

A threshold value γ can be specified. The pair of values with a similarity degree equal or higher than γ are considered to be equal [163]. A Relational Calculus is proposed by [164] for this model. Shenoi and Melton [165] proposed a similar model which works with *proximity relations* instead of similarity relationships. The reflexive and symmetric properties hold in a proximity relation, nevertheless the transitive property does not necessarily holds [162].

2.4.1.2. Prade & Testemale

In his model [43, 44], the knowledge about an attribute for a given entity is represented by means of a normalized possibility distribution. Therefore it is represented incomplete or uncertain information in the database.

A possibility distribution π_A models the available knowledge about the value x takes for the attribute $A \in D$. A special constant e is defined for the case when the value is not applicable. In other words, the possibility distribution can be defined as follows.

$$\pi_A : D \cup \{e\} \mapsto [0, 1] \tag{2.18}$$

The model provides seven different data types:

1. The data are a crisp and known value c.

$$\begin{aligned} \pi_A(e) &= 0 \\ \pi_A(c) &= 1 \\ \pi_A(d) &= 0, \forall d \in D \wedge d \neq c \end{aligned} \tag{2.19}$$

2. The data are unknown but applicable.

$$\begin{aligned} \pi_A(e) &= 0 \\ \pi_A(d) &= 1, \forall d \in D \end{aligned} \tag{2.20}$$

3. The data are not applicable.

$$\begin{aligned} \pi_A(e) &= 1 \\ \pi_A(d) &= 0, \forall d \in D \end{aligned} \tag{2.21}$$

4. There is completely lack of knowledge about the data.

$$\pi_A(d) = 1, \forall d \in D \cup \{e\} \tag{2.22}$$

5. The data is known to be in a range between $[m, n]$.

$$\begin{aligned} \pi_A(e) &= 0 \\ \pi_A(d) &= \begin{cases} 1 & \text{if } d \in [m, n] \subseteq D \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{2.23}$$

6. The knowledge about the data is modelled by a possibility distribution μ_a.

$$\begin{aligned} \pi_{A)}(e) &= 0 \\ \pi_A(d) &= \mu_a(d) \forall d \in D \end{aligned} \tag{2.24}$$

7. The knowledge about the data is modelled by a possibility distribution μ_a. The possibility that the data are not applicable is given by λ.

$$\begin{aligned} \pi_A(e) &= \lambda \\ \pi_A(d) &= \mu_a(d) \forall d \in D \end{aligned} \tag{2.25}$$

2.4.1.3. Umano & Fukami

The main difference between this model [55, 158, 166, 167] and Prade-Testemale is the representation of non-applicable information. Three special values are introduced:

- UNKNOWN: Any value for the attribute in the domain is equally possible. The representation in this model is the following.

$$\pi_A(d) = 1 \forall d \in D \tag{2.26}$$

- UNDEFINED: It is not possible for the attribute to take any value in the domain. The representation in this model is the following.

$$\pi_A(d) = 0 \forall d \in D \tag{2.27}$$

- NULL: The absence of knowledge about the value that the attribute takes is total. This is represented in the model as follows.

$$\text{NULL } = \{1/\text{Unknown}, 1/\text{Undefined}\} \tag{2.28}$$

The main data types are compared with the data types defined in the Prade-Testemale model.

1. The data are a crisp and known value c.

$$\pi_A(d) = \{1/c\}_P \tag{2.29}$$

2. The data are unknown but applicable. See equation (2.26).
3. The data are not applicable. See equation (2.27).
4. There is completely lack of knowledge about the data. See equation (2.28).
5. The data is known to be in a range between $[m, n]$.

$$\pi_A(e) = 0 \tag{2.30}$$
$$\pi_A(d) = \begin{cases} 1 & \text{if } d \in [m, n] \subseteq D \\ 0 & \text{otherwise.} \end{cases}$$

6. The knowledge about the data is modelled by a possibility distribution π_a.

$$\pi_A(d) = \mu_a(d) \forall d \in D \tag{2.31}$$

7. The knowledge about the data is modelled by a possibility distribution π_a. The possibility that the data are not applicable is given by λ. This data type is not representable in the model.

This model was extended by Umano and Fukami [167] to represent and handle ambiguous data. A possibility distribution can be stored in the attribute value. Moreover, the degree of membership for each tuple with respect to the relation is given by a normalized possibility distribution. A membership function μ_R is defined for a fuzzy relationship R with m attributes.

$$\mu_R : \mathcal{P}(U_1) \times \mathcal{P}(U_2) \times \ldots \times \mathcal{P}(U_m) \mapsto \mathcal{P}([0, 1]) \tag{2.32}$$

Where U_j with $j = 1, \ldots, m$ is the Universe for each attribute A_j of the relation R. Two kinds of imprecision can be managed by the system:

1. Imprecision in the attribute value: The value for an attribute may be given by a possibility distribution.
2. Imprecision in the relationship: Each tuple has a membership value with respect to the relationship.

2.4.1.4. Zemankova & Kaendel

In this model [40,168], the information is represented like in the previous presented models, but it offers a language to manipulate data as well as to obtain the relations between possibility and certainty. The model consists on three elements:

1. A value database (VDB) where the data are organized in the same way as the previous database models.
2. An explicative database, where the meta data for fuzzy data and fuzzy relationships are stored.
3. A translation rule set in order to handle adjectives for data and relationships.

The querying process is similar to the Prade-Testemale model. But in this case, instead of providing a necessity measure, they provide a certainty measure. Therefore, the possibility $p_A(F)$ and certainty measures $c_A(F)$ for the fuzzy set F with respect to the attribute value A is defined as follows.

$$\begin{aligned} p_A(F) &= \sup_{u \in D} \{\mu_F(u) \cdot \pi_A(u)\} \\ c_A(F) &= \max_{u \in D} \{0, \inf \{\mu_F(u) \cdot \pi_A(u)\}\} \end{aligned} \tag{2.33}$$

In [40], the selection operators are defined. A similarity relation Θ is defined on D and therefore any other comparison relationship can be defined on it. For example, the relation "bigger than" is defined as follows.

$$\mu_{\text{bigger\_than}}(x, y) = \begin{cases} 1 - 0.5 \cdot s(x, y) & \text{if } x \geq y \\ 0.5 \cdot s(x, y) & \text{if } x < y \end{cases} \tag{2.34}$$

The corresponding possibility and certainty measures are computed by using the equations (2.33).

2.4.1.5. GEFRED by Medina, Vila & Pons

This model [159] is the integration of the main fuzzy database models. GEFRED defines the concept *Generalized Fuzzy Domain*, which is defined as follows.

Definition 31. Generalized Fuzzy Domain.
Consider the powerset $\tilde{\mathcal{P}}(U)$ of all the possibility distributions defined on the Universe U, which include both Unknown and Undefined data types (as given by Umano and Fukami in equations (2.26)-(2.27)). A generalized fuzzy domain D is defined as follows:

$$D \subseteq \tilde{\mathcal{P}}(U) \cup \text{NULL} \tag{2.35}$$

Where the Null constant is given by equation (2.28).

The following data types may be defined in the model:

1. A scalar value. For example, the attribute eye color = blue.

2. A number. For example, the attribute price = 20.
3. A discrete set of scalar values. For example, eye color = {blue, brown}
4. A numeric set of values. For example price = {20, 25}.
5. A possibility distribution over a discrete scalar domain. For example, eye color = {1/blue, 0.1/brown }.
6. A possibility distribution over a numeric domain, a fuzzy set domain or linguistic labels. For example, price = {0.3/20, 1/25}.
7. A number in the unit interval $[0, 1]$. The number represents the membership degree for an attribute in a given tuple. For example Good = 0.8.
8. UNKNOWN: Any value for the attribute in the domain is equally possible.
9. UNDEFINED: It is not possible for the attribute to take any value in the domain.
10. NULL: The absence of knowledge about the value that the attribute takes is total.

Then, a generalized fuzzy relation can be defined.

Definition 32. Generalized Fuzzy Relation.
A generalized fuzzy relation $R = (\mathcal{H}, \mathcal{B})$ is given by two elements. A header $\mathcal{H}$ and a body $\mathcal{B}$ which are defined as follows.

- The header $\mathcal{H}$ is a set of triplets attribute-domain-compatibility:

$$\mathcal{H} = \{(A_1 : D_1 [, C_1]) , \ldots , (A_n : D_n [, C_n])\} \tag{2.36}$$

  Where each attribute A_j has a corresponding fuzzy domain D_j with $j = 0, \ldots , n$. The compatibility attribute is an optional value in the interval $[0, 1]$ and it will be used when an operation is performed.

- The body $\mathcal{B}$ of the relation is the set of tuples where each tuple is a set of triplets.

$$\mathcal{B} = \left\{ \left(A_1 : \tilde{d_{i1}}, [c_{i1}] \right) , \ldots , \left(A_n : \tilde{d_{in}}, [c_{in}] \right) \right\} \tag{2.37}$$

  Where $\tilde{d_{ij}}$ is the domain value for the tuple i on the attribute A_j and c_{ij} is the corresponding compatibility degree.

Given a generalized fuzzy relationship R, the following elements can be defined on it.

Definition 33. Value and compatibility component.
Consider a generalized fuzzy relation R given by the following expression.

$$R = \begin{cases} \mathcal{H} = \{(A_1 : D_1 [, C_1]) , \ldots , (A_n : D_n [, C_n])\} \\ \mathcal{B} = \left\{ \left(A_1 : \tilde{d_{i1}}, [c_{i1}] \right) , \ldots , \left(A_n : \tilde{d_{in}}, [c_{in}] \right) \right\} \end{cases} \tag{2.38}$$

with $i = 1, \ldots , m$ the number of tuples in the relation. The following elements can be defined.

- The value Component R^v of a fuzzy generalized relation is given by the value components of both header and body of the relation.

$$R^v = \begin{cases} \mathcal{H}^v = \{(A_1 : D_1), \ldots, (A_n : D_n)\} \\ \mathcal{B}^v = \left\{ \left(A_1 : \tilde{d_{i1}}\right), \ldots, \left(A_n : \tilde{d_{in}}\right) \right\} \end{cases} \tag{2.39}$$

- Compatibility component R^c of a fuzzy generalized relation is given by the compatibility components of both header and body of the relation.

$$R^c = \begin{cases} \mathcal{H}^c = \{[C_1], \ldots, [C_n]\} \\ \mathcal{B}^c = \{[c_{i1}], \ldots, [c_{in}]\} \end{cases} \tag{2.40}$$

The comparison operators have to be re-defined.

Definition 34. Extended comparison operator.
Let U be the Universe, we will say that θ is an extended comparison operator to any fuzzy relation defined on U.

$$\begin{aligned} \theta : U \times U &\mapsto [0,1] \\ \theta(u_i, u_j) &\mapsto [0,1] \end{aligned} \tag{2.41}$$

Where $u_i, u_j \in U$.

Definition 35. Generalized extended fuzzy comparator.
Let U be the universe and D the generalized fuzzy domain and an extended comparator θ is defined on U. Then, the function Θ^θ is defined as follows:

$$\begin{aligned} \Theta^\theta : D \times D &\mapsto [0,1] \\ \Theta^\theta(\tilde{d_1}, \tilde{d_1}) &\in [0,1] \end{aligned} \tag{2.42}$$

We will say that Θ^θ is a generalized extended fuzzy comparator on D if the following property holds.

$$\Theta^\theta(\tilde{d_1}, \tilde{d_2}) = \theta(d_1, d_2) \, \forall d_1, d_2 \in U \tag{2.43}$$

Where $\tilde{d_1}, \tilde{d_2}$ are the possibility distributions $1/d_1$, $1/d_2$ respectively.

GEFRED defines a generalized fuzzy relational algebra. The operators for the union, intersection, difference, Cartesian product, projection, selection, join and division are defined. In the following, we will introduce the Cartesian product, the projection and the selection operators in GEFRED.

Definition 36. Generalized Cartesian product.
Let R and R' be two generalized fuzzy relations given as follows.

$$R = \begin{cases} \mathcal{H} = \{(A_1 : D_1 [, C_1]), \ldots, (A_n : D_n [, C_n])\} \\ \mathcal{B} = \left\{ \left(A_1 : \tilde{d_{i1}} [, c_{i1}]\right), \ldots, \left(A_n : \tilde{d_{in}} [, c_{in}]\right) \right\} \end{cases}$$

$$R' = \begin{cases} \mathcal{H}' = \{(A'_1 : D'_1 [, C'_1]), \ldots, (A'_{n'} : D'_{n'} [, C'_{n'}])\} \\ \mathcal{B}' = \left\{ \left(A'_1 : \tilde{d'_{k1}} [, c'_{k1}]\right), \ldots, \left(A'_{n'} : \tilde{d'_{kn'}} [, c'_{kn'}]\right) \right\} \end{cases}$$

where $i = 1, \ldots, m$ and $k = 1, \ldots, m'$. The values m and m' are respectively the cardinalities of the relations and n, n' are respectively the arities of the relationships. The generalized Cartesian product is defined as follows.

$$R \times R' = \begin{cases} \mathcal{H}_{\times} & = \mathcal{H} \cup \mathcal{H}' \\ \mathcal{B}_{\times} & = \mathcal{B} \times \mathcal{B}' \end{cases} \tag{2.44}$$

Definition 37. Generalized fuzzy projection.
Let R be a generalized fuzzy relation as shown by equation (2.38), and X a subset of $\mathcal{H}$ notated as follows.

$$X \subseteq \mathcal{H}, X = \{(A_s : D_s [, C_{s'}]) : s \in S, s' \in S'; S, S' \subseteq \{1, \ldots, n\}\}$$

The generalized fuzzy projection is defined by the following formula.

$$\mathcal{P}(R; X) = \begin{cases} \mathcal{H}_{\mathcal{P}} & = X \\ \mathcal{B}_{\mathcal{P}} & = \left\{ \left(A_s : d_{is}, \tilde{[}, c_{is'}] \right) \right\} \end{cases} \tag{2.45}$$

Where $s \in S, s' \in S'$ and $S, S' \subseteq \{1, \ldots, n\}$.

Definition 38. Generalized fuzzy selection.
Let R be a generalized fuzzy relation as shown by equation (2.38), $\tilde{a} \in D$ a constant, Θ^{θ} a generalized fuzzy comparator and $\gamma \in [0, 1]$ a fulfilment threshold. Then, the generalized fuzzy selection on R by using the condition in Θ^{θ} composed with $\tilde{a}$ and the attribute A_k ($k \in \{0, \ldots, n\}$) and qualified by γ is given by the following formula:

$$S\left(R; \Theta^{\theta}\left(A_k, \tilde{a}\right) \geq \gamma\right) = \begin{cases} \mathcal{H}_S & = \{(A_1 : D_a [, C_1]), \ldots, (A_n : D_n [, C_n])\} \\ \mathcal{B}_S & = \left\{ \left(A_1 : \tilde{d_{r1}} [, c_{r1}] \right), \ldots, \left(A_n : d_{rn} \tilde{[}, c_{rn}] \right) \right\} \end{cases} \tag{2.46}$$

with

$$c'_{rk} = \Theta^{\theta}\left(\tilde{d_{rk}}, \tilde{a}\right) \geq \gamma \tag{2.47}$$

where $r = 1, \ldots, m'$ and m' is the cardinality of the selection.

In the following, we will briefly introduce the Fuzzy Interface for Relational Systems (FIRST, [54]) which is based on the GEFRED model. First of all, we will deal with the representation of imperfect information and then, we will deal with the flexible querying.

- **Representation** The representation of imperfect information. FIRST defines three fuzzy data types:
  - *Type 1*: Numerical data with extended querying facilities. The attributes with this data type, can be queried by using the generalized fuzzy comparators introduced before.
  - *Type 2*: Numerical data given by a possibility distribution. There are five subtypes (Crisp, Label, Interval, Approximate and Trapezoidal) which differ in the shape of the distribution and the way the possibility distribution is stored. For example, consider that the price for a book is not precisely known. Then, the average price is stored as shown in Figure 2.9.

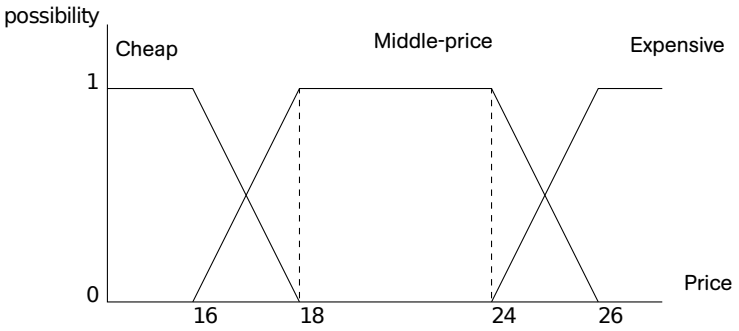


Figure 2.9: Possibility distribution associated to the Price attribute. There are three possibility distributions here, each one correspond with a label; cheap, middle-price and expensive respectively.

- *Type 3*: Data without an underlying numerical domain. In order to compare the elements in the domain, an equivalence table is required. The only applicable operator is the fuzzy equality. For example, consider a restaurant database. Each restaurant has a type of cuisine. It is possible for the user to store the similarity among the different types of cuisine in the database. Table 2.10 shows four types of cuisine (Pizzeria, Hamburger bar, Chinese, Fast Food). A similarity degree in the interval $[0, 1]$ is associated with each pair of types. For example, fast food restaurant and a hamburger bar are considered to be similar with a degree of 0.8. Then, if the user is looking for fast food restaurants, the database would also return some hamburger bars as the result of the query.

	C_1	C_2	C_3	C_4
C_1	1	0	0	.6
C_2	0	1	0	.8
C_3	0	0	1	.4
C_4	.3	.6	.4	1

Table 2.10: Example of a similarity relationship in a non-ordered domain, like the type of cuisine. C_1 = Pizzeria, C_2 = Hamburger bar , C_3 = Chinese, C_4 = Fast Food.

- **Querying** The Fuzzy SQL (FSQL) language is an extension of the SQL standard with support for flexible querying. The fuzzy operators defined in table 2.11 are implemented. The following elements are introduced or modified in FSQL:

  - Fuzzy operators, as shown in Table 2.11. Both possibility and necessity operators are defined and shown in table 2.11. The syntax is the following.

    ```
    <f_attribute> FOperator <f_attribute_or_value>
    ```

For example, we want to know if the price of a book is middle-price, as shown in Figure 2.9. Then the query would be as follows.

```
book.price FEQ $[16,18,24,26];
```

- Linguistic labels. A label is usually associated with a type 2 or type 3 fuzzy type. For example, consider the previous example. We have labels associated to each possibility distribution, then the query would be the following.

  ```
  book.price FEQ $middle_price;
  ```

- Threshold (THOLD): The threshold is a value in the unit interval $[0, 1]$, denoting the minimum global degree that a tuple should fulfill to be selected.

  ```
  <simple_condition> THOLD t;
  ```

  For example, we want to know if the price of a book is middle-price, but we want only the books with a fulfilment degree about 0.8 or more. The query would be as follows.

  ```
  book.price FEQ $[16,18,24,26] THOLD .8;
  ```

- Compatibility degree (CDEG): it is a function that shows the compatibility degree for a tuple or an attribute. For example, in order to show the compatibility degree in a query looking for the books with middle price, the FSQL sentence will be the following.

  ```
  SELECT book.%, CDEG(*)
  FROM book WHERE book.price FEQ $[16,18,24,26];
  ```

- Wildcard (%): When it is specified in the query, the compatibility degree for each fuzzy attribute in the WHERE clause is shown.

- IS: The IS clause is modified in the following way:

  ```
  <fuzzy attribute> IS [NOT]
  (UNKNOWN | UNDEFINED | NULL)
  ```

- Fuzzy quantifiers allow to build complex queries that require to make some calculations about the number of tuples that fulfil a given condition. Similarly to the classical model, there are two main types:

  - Relative quantifiers rely on two quantities and they are usually given in natural language, e.g., "Most of the tuples ...", "The minority of the tuples ...", "Approximately the half of the tuples...". Two values are computed. First the number of tuples that fulfil the query and second, the total number of tuples. Then, an averaging operator is used to compute the fulfilment of the query.

    ```
    $Quantifier FUZZY[r] '(' fuzzy_condition ')'
    THOLD t;
    ```

    For example, in a basketball player database, "Most of the players from team A are very good".

Possibility	Necessity	Possibly / Necessarily
FEQ	NFEQ	Fuzzy $=$
FGT	NFGT	Fuzzy $>$
FGEQ	NFGEQ	Fuzzy $\geq$
FLT	NFLT	Fuzzy.$<$
FLEQ	NFLEQ	Fuzzy $\leq$
MGT	NMGT	Much $>$
MLT	NMLT	Much $<$

Table 2.11: Fuzzy operators

- Absolute quantifiers rely on one quantity. They are related with the number of tuples in the resultset of a query. For example, in natural language, "big", "small", "some", "all", ..., etc. The syntax for these operator is given as follows.

  ```
  $Quantifier FUZZY[r] '(' fuzzy_condition1 ')'
  ARE '(' fuzzy_condition2 ')' THOLD t;
  ```

  For example, in a basketball player database we could combine the two types of quantifiers in a sentence like "Most of the players from team A which are tall, are also very good".

ID	Name	α	β	γ	δ
1	Amadeus	16	18	20	22
2	Grotta Mare	10	15	20	25
3	Saffron	17	20	25	30
4	Castaneda	20	30	30	35

Table 2.12: Example of a restaurant database. The field ID is the restaurant identifier, the primary key. The field name is the name of the restaurant and the fields $[\alpha, \beta, \gamma, \delta]$ are the average price in euros for the restaurant, stored as a trapezoidal possibility distribution.

Example 12. Consider a restaurant database in Table 2.12. A restaurant is identified by a number, ID, which is the primary key. The name of the restaurant and the average price (as a trapezoidal distribution) are stored. The user wants to obtain a list of cheap restaurants. Then, the user defines 'cheap' as a trapezoidal possibility distribution given by $[0, 0, 16, 18]$ (see Figure 2.9). Then, the query, written in FuzzySQL is:

```
SELECT CDEG(*), Restaurants.%
FROM Restaurants
WHERE Price FEQ $[0,0,16,18];
```

The fuzzy equality operator is defined more formally. Consider U to be an underlying domain. Let p_1 and p_2 be two attributes of a fuzzy type, and π_{p_1}, π_{p_2}

be the possibility distributions associated to each attribute respectively. Then the fuzzy equals operator is defined formally as follows.

$$\text{FEQ}\ (p_1, p_2) = \sup_{d \in U} \min \left(\pi_{p_1}(d), \pi_{p_2}(d) \right) \tag{2.48}$$

CDEG(*)	**ID**	Name
1.0	2	Grotta Mare
0.5	1	Amadeus
0.2	3	Saffron
0.0	4	Castaneda

Table 2.13: Example of a restaurant database. Resultset for the query: *'The user wants to obtain all the cheap restaurants'*. The resultset is ordered by the compatibility degree.

Then, the resultset ordered by compatibility degree, is presented to the user. See Table 2.13.

2.5. Conclusions

In this chapter, we have studied the time modelling in Information Systems. The main concepts when dealing with temporal information have been presented. A *chronon* [73] is the smallest unit of time that can be represented in a system. Next to that, we have studied the organization of the time in hierarchical structures such as granularities and Calendars. Several authors proposed methods for handling the changes between different granularities.

We have presented an example of temporal domain, the Julian Day Number (JDN) [93] which consist on the representation of a date as a number. This representation is of particular interest since some of the major vendors of database systems provide an internal representation of the time in this format.

A time instant can be represented as a time point either as a time interval. Due to the limitation (discretization) in the representation of time in an Information System, a common approach is to model a time instant as a time interval. The relations between two crisp time intervals have been studied by Allen [9]. Only thirteen relations are available between two crisp time intervals.

Nevertheless, humans manage and handle temporal information by using temporal notions which may contain imperfections. Several studies have been done to understand time in language [2–4]. In the field of artificial intelligence, some efforts have been done to model and reason with imperfect temporal information [5, 6, 169, 170].

In order to classify imperfections in temporal information, we studied first the sources of imperfections in an Information System [99, 100]. Then, we provide several classifications for imperfect temporal information.

Once we have classified the temporal information, we provide an overview of the main theoretical frameworks to deal with imperfections in temporal information. Probability theory is used when a probability distribution is associated to the temporal expression. For example, "The package will arrive tomorrow morning with a probability of 0.8". Possibility theory is used when a possibility distribution can be associated to the temporal expression. Rough set theory has been used to represent uncertainty in time intervals.

In order to represent and handle imperfect time intervals, we choose the possibility theory framework. This choice is motivated by the facilities of possibility theory to model complex relations between time intervals as well as the support in fuzzy relational databases.

A brief overview of temporal database approaches is given. The models are classified depending on the kind of temporal information that is managed by the database. Three kind of temporal data types are studied; valid-time is the time when a fact is recorded in a database. For example, a credit card transaction has a transaction-time which is a timestamp indicating the time when the transaction was done. Valid time is the time when a fact is valid in the modelled reality. For example, in an employee's database modelling the data about the contracts of the employees, the valid-time for a contract is the time when the contract is valid. Decision time is a time when a decision for an event was made. For example, in an employee's database, decision-time is a timestamp indicating when the decision to hire an employee was done.

There are two main issues that have to be addressed when dealing with time-

dependent information in a database. First, the primary key has to be extended to allow different versions of the same object. Second, a consistence mechanism has to be provided by the database which is usually provided by re-defining the data manipulation language (DML).

There are some commercial database management systems with temporal support. Among them, Oracle Workspace Manager stands out because of its complete temporal support. Nevertheless, none of the commercial systems provide support for imperfect temporal information.

Some basic concepts about fuzzy modelling by using fuzzy set theory were introduced. Then we studied some of the most popular fuzzy database proposals. The main contributions for the fuzzy databases are briefly studied. The GEFRED model [159] is the integration of the main fuzzy database models. We will build a temporal database model to deal with imperfect temporal information on the top of it.

The next chapter is devoted to present a theoretical model for the possibilistic valid-time approach on the top of the GEFRED model.

3

A Possibilistic Valid-time Model

The contents of this chapter have been partially published on:

- J. E. Pons, N. Marín, O. Pons Capote, C. Billiet, and G. de Tre, "A relational model for the possibilistic valid-time approach," *International Journal of Computational Intelligence Systems*, vol. 5, no. 6, pp. 1068–1088, 2012.
- A. Bronselaer, J. E. Pons, G. De Tré, and O. Pons, "Possibilistic evaluation of sets," *Int. J. Uncertainty Fuzziness Knowlege-Based Syst.*, vol. 21, no. 3, pp. 325–346, 2013.
- J. E. Pons, C. Billiet, O. Pons Capote, and G. Tré, "A possibilistic valid-time model," in *Advances on Computational Intelligence* (S. Greco, B. Bouchon-Meunier, G. Coletti, M. Fedrizzi, B. Matarazzo, and R. Yager, eds.), vol. 297 of *Communications in Computer and Information Science*, pp. 420–429, Springer Berlin Heidelberg, 2012.
- C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, "Evaluating possibilistic valid-time queries," in *Advances on Computational Intelligence* (S. Greco, B. Bouchon-Meunier, G. Coletti, M. Fedrizzi, B. Matarazzo, and R. Yager, eds.), vol. 297 of *Communications in Computer and Information Science*, pp. 410–419, Springer Berlin Heidelberg, 2012.
- J. Pons, A. Bronselaer, O. Pons, and G. de Tre, "Possibilistic evaluation of fuzzy temporal intervals," in *Actas del XVI congreso Español sobre Tecnologías y Lógica Fuzzy* (Valladolid, Spain), february 2012.

Contents

In this chapter we will introduce a new valid-time model to represent and handle ill-known temporal intervals.

We will introduce some background concepts about possibility theory, which is the mathematical tool we use for modelling imperfections. Next we introduce our possibilistic framework, which is extended in depth in Appendix A. Furthermore, we explain the representation of time within the possibilistic framework.

A crisp model for temporal relational databases is presented. In the next section, this model is extended by using the possibilistic framework developed before.

3.1. Preliminaries

In this section, we introduce some basic concepts concerning possibilistic variables and fuzzy numbers and intervals. Then, the framework of set evaluation by ill-known constraints [25] is explained. The section concludes with a brief introduction to temporal databases.

3.1.1. Possibility Theory

In the past century, several theories have been proposed that deal with the modelling, processing and inference of uncertainty. The most popular of these theories is perhaps probability theory [34, 35], which has many important applications in statistical testing and predictive models, amongst others. Despite the huge importance of probability theory, it has been argued by several authors in the past decades that the purpose and applicability of probability theory is restricted. More specific, it has been shown that probability theory deals with uncertainty caused by *variability* in the outcome of experiments [38]. For example, if a coin is tossed, this can be regarded as an experiment and the outcome is variable in that sense that tossing the coin repeatedly will result in different outcomes. The uncertainty about the answer to the question "What side of the coin will be shown after the toss?" is caused by the fact that the experiment has two possible outcomes. It is however well known that there is another cause of uncertainty that can not be captured by probability theory: incomplete knowledge [155, 156]. For example, the uncertainty about the question "What is the birthday of John?" is clearly not caused by some variability in the outcome of an experiment, but it is caused by a lack of knowledge about a person named John.

In order to cope with this alternative cause of uncertainty, the theory of possibility has been developed and it has been argued that it is a theory compatible to probability theory. Possibility theory was first defined by Zadeh [36], although some earlier works already provide an informal treatment of possibility theory [2, 171]. For a concise and formal treatment of possibility theory, the authors refer to [172–174]. Perhaps the most important contribution of Zadeh [36] is the establishment of a connection between possibility theory on the one hand and fuzzy sets [17] on the other hand. This connection contributes to the observation that the membership degree of a fuzzy set can be assigned three semantics: degree of similarity, degree of preference and degree of uncertainty [175].

Informally, a theory of uncertainty considers a set of uncertain events and tries to quantify the confidence about the occurence of those events. Such quantification of

confidence is achieved by means of a *confidence measure* [176].

Definition 39. Confidence measure.
Consider a set of outcomes Ω. Let $\mathcal{P}(\Omega)$ denote the powerset of Ω and let A and B be elements of $\mathcal{P}(\Omega)$. A *confidence measure on* Ω is defined by a function

$$g : \mathcal{P}(\Omega) \rightarrow [0, 1] \tag{3.1}$$

that satisfies

$$g(\emptyset) = 0 \tag{3.2}$$

$$g(\Omega) = 1 \tag{3.3}$$

$$A \subseteq B \Rightarrow g(A) \leq g(B) \tag{3.4}$$

Both possibility and necessity measures are special cases of confidence measures.

Definition 40. Possibility measure.
Consider a confidence measure Π on a set of outcomes Ω. Let J be a countable index set and let $\{A_j | j \in J \wedge A_j \subseteq \Omega\}$ be a family of elements of $\mathcal{P}(\Omega)$. Π is a *possibility measure on* Ω if it satisfies:

$$\Pi\left(\bigcup_{j \in J} A_j\right) = \sup_{j \in J} \Pi(A_j) \tag{3.5}$$

In this work, the interpretation is as follows. The possibility of an event expresses how plausible the occurrence of the event seems to an observer of the experiment, given the (partial) knowledge of the observer about the experiment.

Information on the possibility of distinct elements of the universe of discourse Ω can now be given by a *possibility distribution* π on Ω, defined by:

Definition 41. Possibility distribution.
Consider a possibility measure Π on Ω. A *possibility distribution* π on Ω underlying the possibility measure Π is a function defined by:

$$\pi : \Omega \rightarrow [0, 1] : \pi(u) = \Pi(\{u\}) \tag{3.6}$$

Definition 42. Necessity measure.
Consider a confidence measure N on Ω. Let J be a countable index set and let $\{A_j | j \in J \wedge A_j \subseteq \Omega\}$ be a family of elements of $\mathcal{P}(\Omega)$. N is a *necessity measure* on Ω if it satisfies:

$$N\left(\bigcap_{j \in J} A_j\right) = \inf_{j \in J} N(A_j) \tag{3.7}$$

In this work, the interpretation is as follows. The necessity of an event expresses how necessary the occurrence of the event seems to an observer of the experiment, given the (partial) knowledge of the observer about the experiment.

Possibility and necessity measures are dual in the sense that:

$$\forall A \subseteq \Omega : N(A) = 1 - \Pi(\bar{A}) \tag{3.8}$$

That is, the degree to which an event is necessary is the extent to every other possible event is not plausible.

3.1.2. Possibilistic Variables

Informally speaking, a possibilistic variable is a variable taking one value, but for some reason its value is unknown. Then, a possibility distribution is defined to model the knowledge available about the set of possible values for such a variable. Consider for example the height of a person. The exact value for a given person might not be precissely known, but it could be specified by a possibility distribution. Possibilistic variables rely on possibility theory [151] and are defined as follows [25].

Definition 43. Possibilistic variable.
A possibilistic variable X over a universe U is defined as a variable taking exactly one value in U, but for which this value is (partially) unknown. A possibility distribution π_X on U models the available knowledge about the value X takes; for each $u \in U$, $\pi_X(u)$ represents the possibility that X takes the value u.

In this work, this possibility is interpreted as a measure of how plausible it is that X takes the value u, given the available (partial) knowledge about the value X takes. The exact value a possibilistic variable takes, which is (partially) unknown, is called an *ill-known value* in this work [151].

When a possibilistic variable is defined on the powerset $\mathcal{P}(U)$ of some universe U, the unique value the variable takes will be a crisp set and its possibility distribution on the powerset $\mathcal{P}(U)$ will describe the possibility of each crisp subset of U to be the value the variable takes. This exact value (a crisp set) the variable takes, is now called an *ill-known set* [151].

Note that while a single ill-known value refers to one (partially) unknown value, an ill-known set is a crisp set but, for some reason, (partially) unknown.

A specific application of possibilistic variables is obtained when the universe under consideration is the set of Boolean values $\mathbb{B} = \{T, F\}$. Indeed, any Boolean proposition p takes just one value in $\mathbb{B}$. If the knowledge about which value this proposition p will take is given by a possibility distribution π_p; the proposition can be seen as a possibilistic variable. The possibility and necessity that p are T (the proposition holds) demand more attention. This possibility and necessity is noted here as:

$$\text{Possibility that } p = T \text{ (p holds):} \qquad Pos(p) = \pi_p(T) \tag{3.9}$$

$$\text{Necessity that } p = T \text{ (p holds):} \qquad Nec(p) = 1 - \pi_p(F) \tag{3.10}$$

Here, equation (3.9) denotes the possibility that $p = T$ and the proposition holds, while equation (3.10) denotes the necessity that $p = T$ and the proposition holds.

This work deals with ill-known intervals. These are ill-known sets, defined and represented via a starting and an ending point that, in turn are ill-known values. The elements of the set are the values between the starting and ending points. A closed ill-known interval with a starting point defined by a possibilistic variable X and an ending point defined by a possibilistic variable Y is noted here $[X, Y]$.

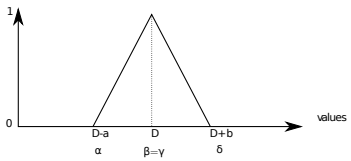


Figure 3.1: Example of fuzzy number.

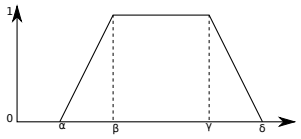


Figure 3.2: Example of fuzzy interval.

3.1.3. Fuzzy Numbers and Fuzzy Intervals

Among others, Dubois and Prade [177] use fuzzy sets [17] to define a *fuzzy interval*.

Definition 44. Fuzzy interval.
A fuzzy interval is a fuzzy set M, defined by a membership function μ_M on the set of real numbers $\mathbb{R}$ such that:

$$\begin{aligned} \mu_M : \mathbb{R} &\rightarrow [0,1] \\ \forall (u,v) \in \mathbb{R}^2 : \forall w \in [u,v] &: \\ \mu_M(w) &\geq \min(\mu_M(u), \mu_M(v)) \\ \exists m \in \mathbb{R} : \mu_M(m) &= 1 \end{aligned} \tag{3.11}$$

If this modal value m is unique, then M is referred to as a *fuzzy number*.

A simple form of the membership function of a fuzzy interval is a trapezoidal function (Figure 3.2). It can be shown that such a membership function μ_T for a fuzzy interval T is convex and normalized. The values $[\alpha, \beta, \gamma, \delta]$ represent a trapezoidal possibility distribution defined by μ_T:

$$\mu_T : \quad \mathbb{R} \rightarrow [0,1] \tag{3.12}$$

$$\mu_T(x) = \begin{cases} 1 & \text{if } x \in [\beta, \gamma] \\ 0 & \text{if } x > \delta \vee x < \alpha \\ \frac{x-\alpha}{\beta-\alpha} & \text{if } x \in [\alpha, \beta[\\ \frac{\delta-x}{\delta-\gamma} & \text{if } x \in]\gamma, \delta] \end{cases}$$

The representation for a triangular possibility distribution (Figure 3.1) is given by three values $[D, a, b]$. The distribution is obtained from a trapezoidal possibility distribution by replacing the values $[D-a, D, D, D+b]$. Both representations for a triangular possibility distribution are equivalent, but the compact representation $[D, a, b]$ is

preferred and used in this work because the values a and b need to be calculated when computing the membership function for the triangular possibility distribution.

3.1.4. Interval Evaluation by Ill-known Constraints

In this section we will explain first the concept of constraint and then, we will illustrate how to extend it when the constraint is applied on an ill-known value.

Definition 45. Given a universe U, a constraint C on a set $A \subseteq U$ is specified by means of a binary relation $R \subseteq U^2$ and a fixed value $x \in U$, i.e.:

$$C \triangleq (R, x)\,. \tag{3.13}$$

It is said that a set A satisfies the constraint C if and only if:

$$\forall a \in A : (a, x) \in R. \tag{3.14}$$

Definition 45 adheres to the fact that the framework of constraint evaluation is Boolean in nature. In order to make an explicit connection with the Boolean framework, we shall adopt the notation $C(A)$ to indicate a Boolean proposition which is true if A satisfies C and which is false if A fails (i.e. does not satisfy) C.

Example 13. Consider the set of natural numbers $\mathbb{N}$ and consider the constraint $C = (\leq, 3)$, then the set $A = \{1, 2, 3\}$ satisfies constraint C because all elements in A are in relation $\leq$ to the value 3.

Example 13 illustrates that crisp constraints are quite trivial. However, the triviality disappears when the step towards ill-known constraints is made. An ill-known constraint differs from a constraint in the sense that the value in the constraint specification is no longer a crisp value x, but an ill-known value X. This generalization requires a mechanism for checking whether or not a crisp value is in relation to an ill-known value. Such a mechanism is provided by application of Zadeh's Extension Principle. More specific, for any ill-known value X over U and for any binary relation R over U, we have that:

$$\forall u \in U : \mathrm{Pos}((u, X) \in R) = \sup_{(u,w)\in R} \pi_X(w) \tag{3.15}$$

$$\forall u \in U : \mathrm{Nec}((u, X) \in R) = \inf_{(u,w)\notin R} 1 - \pi_X(w). \tag{3.16}$$

In the case of temporal databases, it is very important to know if all points in a crisp interval I reside between the boundaries of an ill-known interval $[X, Y]$. To compute that, the concept of *ill-known constraint* [25] has been introduced.

Definition 46. Ill-known constraint.
Given a universe U, an ill-known constraint C on a set $A \subseteq U$ is specified by means of a binary relation $R \subseteq U^2$ and a fixed ill-known value denoted by V over U, i.e.:

$$C \triangleq (R, V) \tag{3.17}$$

Set A satisfies the constraint if and only if:

$$\forall a \in A : (a, V) \in R \tag{3.18}$$

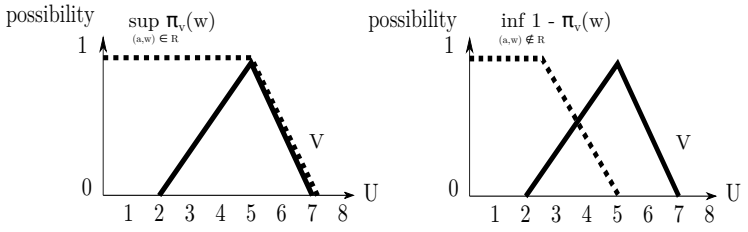


Figure 3.3: Example of the evaluation of the ill-known constraint $C \triangleq (\leq, V), V = [5, 3, 2]$. The left graphic shows the calculation of $\sup_{(a,w)\in R} \pi_V(w)$. The right graphic shows the calculation of $\inf_{(a,w)\notin R} 1 - \pi_V(w)$.

An example of an ill-known constraint is $C_{ex} = (<, V)$. Some set A satisfies C_{ex} if $\forall a \in A : a < V$, given the possibilistic variable V.

The satisfaction of a constraint $C \triangleq (R, V)$ by a set A is still a Boolean matter, but due to the uncertainty about the ill-known value V, it can be uncertain whether C is satisfied by A or not [25]. In fact, this satisfaction now behaves as a proposition. Based on the possibility distribution π_V of V, the possibility and necessity that A satisfies C can be computed. This proposition can thus be seen as a possibilistic variable on $\mathbb{B}$. The required possibility and necessity are (See appendix A):

$$\text{Pos}(A \text{ satisfies } C) = \min_{a\in A} \left(\sup_{(a,w)\in R} \pi_V(w) \right) \tag{3.19}$$

$$\text{Nec}(A \text{ satisfies } C) = \min_{a\in A} \left(\inf_{(a,w)\notin R} 1 - \pi_V(w) \right) \tag{3.20}$$

So far, we have shown how it can be verified whether a crisp set satisfies or not an ill-known constraint. The interval evaluation problem is explained in a more general context in [25].

Example 14. Consider $V = [5, 3, 2]$ and let $C = (\leq, V)$ the ill-known constraint. Then, the evaluation of the possibility and the necessity are obtained from (3.19) and (3.20) respectively. The intermediate calculations for both possibility and necessity measures are shown in Figure 3.3.

$$\text{Pos}(A \text{ satisfies } C) = \min_{a\in A} \left(\sup_{a\leq w} \pi_X(w) \right) \tag{3.21}$$

$$\text{Nec}(A \text{ satisfies } C) = \min_{a\in A} \left(\inf_{a>w} 1 - \pi_X(w) \right) \tag{3.22}$$

It is observed that Boolean combinations of constraints are required. For example, the problem of interval evaluation (explained earlier) requires that all the elements of an interval $[a, b]$ are larger than a value X and, at the same time, smaller than a value Y, which implies that a conjunctive Boolean combination of both constraints must be satisfied. To allow Boolean combinations of constraints, the following definitions are introduced.

Definition 47. Evaluation function.
Consider a universe U, an n-ary vector $\mathbf{C}$ of constraints and a Boolean function $\mathcal{B} : \mathbb{B}^n \to \mathbb{B}$. An evaluation function is defined by:

$$\lambda : \mathcal{P}(U) \to \mathbb{B} : \lambda(A) \mapsto \mathcal{B}\Big(C_1(A), ..., C_n(A)\Big). \tag{3.23}$$

Definition 47 presents a function that evaluates a Boolean combination of some basic constraints. Informally, it states that a set A passes the evaluation made by λ if the evaluation of the Boolean combination of some propositions C_i, (with $i = 1, \ldots, n$) yields T. This crisp definition can be generalized to the case of ill-known constraints.

Definition 48. Ill-known set evaluation.
Consider a universe U, an n-ary vector $\mathbf{C}$ of ill-known constraints and a Boolean function $\mathcal{B} : \mathbb{B}^n \to \mathbb{B}$. The uncertainty about the evaluation of a set A by an evaluation function λ is then given by:

$$\forall A \in \mathcal{P}(U) : \pi_{\lambda(A)} = \widetilde{\mathcal{B}}\Big(\pi_{C_1(A)}, ..., \pi_{C_n(A)}\Big) \tag{3.24}$$

Hereby, $\widetilde{\mathcal{B}}$ is the possibilistic extension of $\mathcal{B}$.

It is well known that any Boolean function $\mathcal{B}$ can be cast to a canonical form [178], requiring only logical conjunction $\wedge$, logical disjunction $\vee$ and logical negation. Therefore, only those connectives will be treated within the scope of this paper. By applying the possibilistic extensions of $\wedge$, $\vee$ and $\neg$, concrete equations are obtained for the calculations of uncertainty about the evaluation of a set by means of an evaluation function λ. In the case of conjunction (i.e., $\mathcal{B} = \wedge$), the inference of uncertainty about the evaluation of a set reduces to:

$$\forall A \in \mathcal{P}(U) : \mathrm{Pos}(\lambda(A)) \quad = \min_{i=1...n} \mathrm{Pos}\left(C_i(A)\right) \tag{3.25}$$
$$\forall A \in \mathcal{P}(U) : \mathrm{Nec}(\lambda(A)) \quad = \min_{i=1...n} \mathrm{Nec}\left(C_i(A)\right). \tag{3.26}$$

In the case of disjunction (i.e. $\mathcal{B} = \vee$), the inference of uncertainty about the evaluation of a set reduces to:

$$\forall A \in \mathcal{P}(U) : \mathrm{Pos}(\lambda(A)) \quad = \max_{i=1...n} \mathrm{Pos}\left(C_i(A)\right) \tag{3.27}$$
$$\forall A \in \mathcal{P}(U) : \mathrm{Nec}(\lambda(A)) \quad = \max_{i=1...n} \mathrm{Nec}\left(C_i(A)\right). \tag{3.28}$$

Note that by using the functions $\min$ and $\max$ here, there is an implicit assumption that the possibilistic variables π_{C_i} are mutual $\min$-dependent in the sense of De Cooman (i.e. non-interactive). For an extensive reading on (in)dependency of possibilistic variables, the reader is referred to [172], [173], [174]. In case of $\neg$, we get:

$$\forall A \in \mathcal{P}(U) : \mathrm{Pos}(\neg\lambda(A)) \quad = 1 - \mathrm{Nec}(\lambda(A)) \tag{3.29}$$
$$\forall A \in \mathcal{P}(U) : \mathrm{Nec}(\neg\lambda(A)) \quad = 1 - \mathrm{Pos}(\lambda(A)). \tag{3.30}$$

Example 15. Consider that we want to check if the crisp interval $I = [j, k]$ is included in $[X, Y]$. In this situation, two ill-known constraints are constructed.

$$C_1 \triangleq (\geq, X) \tag{3.31}$$
$$C_2 \triangleq (\leq, Y) \tag{3.32}$$

To calculate the possibility and necessity concerning a conjunction of constraints, the $\min$ operator can be used. The possibility and necessity of I being included in $[X, Y]$ are now:

$$\text{Pos}(I \text{ satisfies } C_1 \textit{ and } C_2) = \min_{a \in I} \left(\sup_{a \geq w} \pi_X(w), \sup_{a \leq v} \pi_Y(v) \right) \tag{3.33}$$

$$\text{Nec}(I \text{ satisfies } C_1 \textit{ and } C_2) = \min_{a \in I} \left(\inf_{a < w} 1 - \pi_X(w), \inf_{a > v} 1 - \pi_Y(v) \right). \tag{3.34}$$

The conjunction of ill-known constraints is of particular interest in our research. In the following we will present and study the properties of this operation. The conjunctive combination of ill-known constraints evaluated over the same variable is calculated by Eq. (3.25), (3.25). In the following we will define the evaluation and the properties of the conjunctive combination of ill-known constraints over different variables.

Definition 49. Conjunctive combination of ill-known constraints.

Consider a universe U. Let R_1, R_2 be two binary relations in U. Let X_1, X_2 be two fixed ill-known values in U. Let $C_1 = (R_1, X_1)$ and $C_2 = (R_2, X_2)$ be two ill-known constraints.

$$CC \triangleq \{C_1 \wedge C_2\} \tag{3.35}$$

In a more general way, it is possible to define the conjunctive combination of an n-ary vector of constraints:

$$CC \triangleq \{C_1 \wedge \ldots \wedge C_n\} \tag{3.36}$$

Theorem 1. Consider the conjunctive combination CC of any n-ary vector $[C_{1_{Z_1}}, \ldots, C_{n_{Z_n}}]$ of constraints over the sets $Z_1, \ldots, Z_n$. Then, if $\pi_{C_1(Z_1)}, \ldots, \pi_{C_n(Z_n)}$ are convex, then π_{CC} is also convex.

Proof. Let $CC \triangleq \left\{C_{1_{Z_1}} \wedge \ldots \wedge C_{n_{Z_n}}\right\}$. Then:

$$\begin{aligned} \pi_{CC}\left(\omega x_1 + (1-\omega)\, x_2\right) = \min \big(& \pi_{C_1(Z_1)}\left(\omega x_1 + (1-\omega)\, x_2\right), \\ & \ldots, \pi_{C_n(Z_n)}\left(\omega x_1 + (1-\omega)\, x_2\right) \big) \end{aligned} \tag{3.37}$$

Since $\pi_{C_1(Z_1)}, \ldots, \pi_{C_n(Z_n)}$ are convex:

$$\begin{aligned} \pi_{C_1(Z_1)}\left(\omega x_1 + (1-\omega)\, x_2\right) &\geqq \min\left(\pi_{C_1(Z_1)}\left(x_1\right), \pi_{C_1(Z_1)}\left(x_2\right)\right) \\ &\vdots \\ \pi_{C_n(Z_n)}\left(\omega x_1 + (1-\omega)\, x_2\right) &\geqq \min\left(\pi_{C_n(Z_n)}\left(x_1\right), \pi_{C_n(Z_n)}\left(x_2\right)\right) \end{aligned} \tag{3.38}$$

Then, by using equation (3.37):

$$\pi_{CC}\left(\omega x_1 + (1-\omega)\, x_2\right) \geqq \min\big(\min\left(\pi_{C_1(Z_1)}\left(x_1\right), \pi_{C_1(Z_1)}\left(x_2\right)\right), \ldots, \min\left(\pi_{C_n(Z_n)}\left(x_1\right), \pi_{C_n(Z_n)}\left(x_2\right)\right)\big) \tag{3.39}$$

Which is equivalent to the following:

$$\pi_{CC}\left(\omega x_1 + (1-\omega)\, x_2\right) \geqq \min\big(\min\left(\pi_{C_1(Z_1)}\left(x_1\right), \ldots, \pi_{C_n(Z_n)}\left(x_1\right)\right), \ldots, \min\left(\pi_{C_1(Z_1)}\left(x_2\right), \ldots, \pi_{C_n(Z_n)}\left(x_2\right)\right)\big) \tag{3.40}$$

Finally we obtain:

$$\pi_{CC}\left(\omega x_1 + (1-\omega)\, x_2\right) \geqq \min\left(\pi_{CC}\left(x_1\right), \pi_{CC}\left(x_2\right)\right) \tag{3.41}$$

□

Sometimes, an ill-known value might be specified by a convex combination of ill-known constraints. This allows to define ill-known values by means of relationships with respect to other ill-known points.

Definition 50. Ill-known value defined by conjunctive combination of constraints. Consider a universe U, and $CC \triangleq \left\{C_{1_{Z_1}} \wedge \ldots \wedge C_{n_{Z_n}}\right\}$ a conjunctive combination of ill-known constraints over the variables $Z_1, \ldots, Z_n$. The uncertainty about the evaluation of an ill-known value X is given by:

$$X \in \mathcal{P}(U) : \pi_X = \pi_{CC} \tag{3.42}$$

The definition of the ill-known value X with respect to the conjunctive combination of ill-known constraints, is written as:

$$X \triangleq CC \tag{3.43}$$

Note that π_X is convex since π_{CC} is convex as demonstrated in Theorem 1.

Example 16. As an example, consider a historical database containing data about diplomatic medieval documents. The starting and ending dates when a diplomatic document was valid, are not precisely known. Consider now that the time granularity are years. Then $X = [1112, 2, 2]$ is the starting year for the validity of a document. A new diplomatic document was valid in the year $Y = [1118, 2, 1]$. Then, it is possible to obtain Z (the period of time between the starting of both documents) by using a conjunctive combination of constraints.

$$CC = \{C_1\left(>, X\right) \wedge C_2(\leq, Y)\}$$
$$Z = CC$$

Z is a fuzzy interval defined by a trapezoidal shape given by [1112, 1114, 1118, 1119]. Figure 3.4 illustrates the relations among the variables X, Y and Z.

We have seen the main theoretical concepts about ill-known values. Now, we are going to explain the main concepts about the treatment of time and the imperfection related to time in databases. These two preliminary analysis will be the pillars of our proposal in section 3.2.

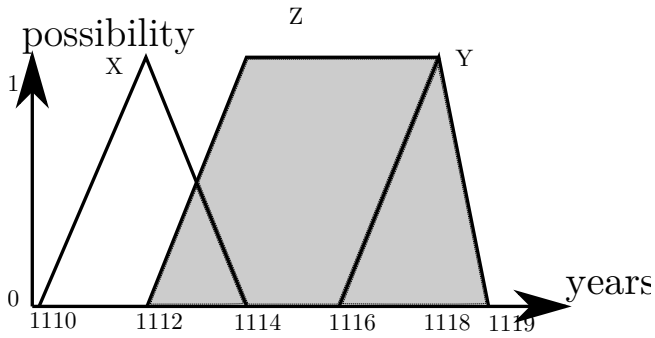


Figure 3.4: Ill-known values X and Y. The grey area represents the ill-known value Z defined by the convex combination of the two ill-known constraints C_1 and C_2.

3.2. Time Representation

This section is devoted to specify the representation of time within the framework of possibility theory. First of all, we will present some approaches in the literature. Next to that we will define the specification for a single ill-known temporal point. Finally, the formal specification and the related constraints are given for an ill-known valid-time interval.

3.2.1. Approaches for the representation

Several proposals for managing uncertain time in a database exist. Some proposals work with rough sets [26], and some others rely on possibility distributions for representing uncertainty in time [24,74,179]. To compare temporal possibility distributions, extensions of the classical Allen's operators [9] are defined in [19, 22, 23, 180].

In order to deal with uncertainty in time intervals, several proposals have been made. Here, two approaches are described: the first one, based on *Fuzzy Validity Periods* [24] and the second one, based on *Possibilistic Valid-time Periods* [181].

Definition 51. A **Fuzzy Validity Period** [24] (*FVP*) is defined as a fuzzy time interval specifying when the data regarding an object are valid. A fuzzy time interval is then the fuzzification of a crisp time interval.

Several options to transform possibility distributions corresponding to the fuzzy starting point and the fuzzy end point into a consistent FVP exist [24], e.g., Fig. 3.5:

- The **convex hull** approach is the most intuitive approach. The resulting FVP is the convex hull of the union of both possibility distributions.
- In the **uncertainty preserving** approach, the amount of uncertainty is maintained at the edges of the possibility distribution representing the FVP [24].

The main feature for the FVP is the optimization for the storage. The compact representation is the result of a conjunctive semantic. The object is valid within all the time points inside the starting and ending points.

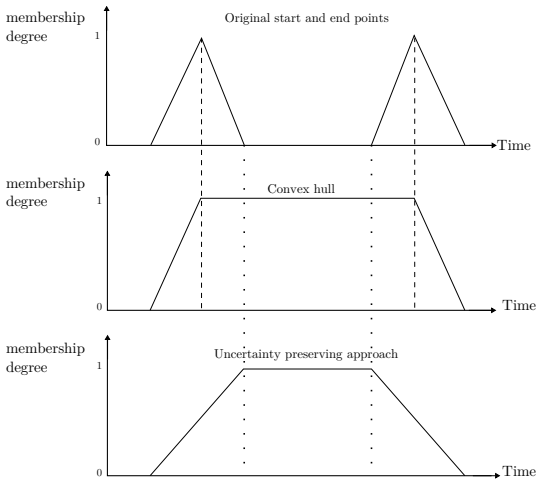


Figure 3.5: Transformation to obtain the FVP. The top graph shows the two triangular possibility distributions. The middle graph shows the convex hull validity period, the bottom one shows the result of the second transformation, which maintains the imprecision.

Definition 52. A **Possibilistic Valid-time Period** (*PVP*) is an ill-known interval of time specifying when the data regarding an object are valid.

Note that the PVP represents only one crisp time interval, but for some reason, it is (partially) unknown.

The main advantage for the PVP is that it preserves all the information for both starting and ending points [25, 182]. Table 3.1 is a comparison between PVP and FVP. The following list defines the items in the comparative:

(1) Domain: The domain of the possibility distribution modelled by the approach.

(2) Implementation of relationships: How to implement a relationship?.

(3) Allen's relations: Are the Allen relations defined?

(4) Storage: The way the data are stored in the database.

(5) Possibility measures: Does the framework provide a possibility measure for any relation between two time intervals?

(6) Necessity measures: Does the framework provide a necessity measure for any relation between the temporal elements?

In the rest of the work, we will work only with PVP to represent valid-time intervals.

3.2.2. Ill-known time point

An ill-known time point X is a single time point that, for some reason, is not fully specified. Note that X has only one possible value but that value is unspecified.

Item	PVP	FVP
(1)	$\mathcal{P}(\mathbb{R})$	$\mathbb{R}$
(2)	Ill-known constraints.	Ad-hoc operators.
(3)	✓	-
(4)	Two distributions one for the starting point and one for the ending point.	Only one distribution.
(5)	✓	If the FVP is obtained by the convex hull transformation, then the possibility measure returns the same value than the possibility measure for the PVP. If the FVP is obtained by the preserving transformation approach, then the possibility measure returns a different result than the possibility measure of the PVP.
(6)	✓	The necessity measure results in an information loss and provides a different value than the necessity measure of the PVP.

Table 3.1: Comparative PVP vs FVP

Definition 53. Ill-known time point.
Consider a time domain $\mathcal{T}$; the uncertainty about the values of an ill-known time point X is given by the possibility distribution π_X:

$$\Pi_X(\{t\}) = \pi_X(t) \in [0,1], t \in \mathcal{T} \tag{3.44}$$

It is also possible to specify an ill-known time point by a convex combination of ill-known constraints, as specified by equation (3.42).

Definition 54. Domain for an ill-known time point.
Consider $\mathcal{P}(\mathcal{T})$ the set of all the possibility distributions over $\mathcal{T}$, and the three fuzzy constants *UNKNOWN* $= \{1/t, \forall t \in \mathcal{T}\}$, *UNDEFINED* $= \{0/t, \forall t \in \mathcal{T}\}$ and *NULL* $= \{1/$ *UNKNOWN*, $1/$ *UNDEFINED* $\}$. The domain for an ill-known time point X is given by:

$$\mathcal{D}(X) = \{\mathcal{P}(\mathcal{T}) \cup \textit{UNKNOWN} \cup \textit{UNDEFINED} \cup \textit{NULL}\} \tag{3.45}$$

3.2.2.1. Data Types

The data type for the representation of an ill-known time point allows the representation of the values shown in Table 3.2

Example 17. Consider a historical database with data from medieval diplomatic documents. The following fields are stored: the digital identifier *ID* which is the primary key and the estimated time when the document was issued (field *Date*). Table 3.3 contains some example data from this database.

Subtype	Value	Representation
1	A single time point	$1/x, x \in \mathcal{T}$
2	A possibility distribution in the numeric domain	A fuzzy number or a fuzzy interval.
3	An unknown value	**UNKNOWN**= $\{1/t, \forall t \in \mathcal{T}\}$
4	An undefined value	**UNDEFINED**= $\{0/t, \forall t \in \mathcal{T}\}$
5	A null value	**NULL** = $\{1/$Unknown, 1/Undefined $\}$

Table 3.2: Values for the time point data type.

ID	Date
23454	Unknown
34563	11/12/1204
12211	$[7/2/1204, 30, 30]$
23455	$[10, 10/6/1204, 20/6/1204, 15]$

Table 3.3: Sample of the historical database

In that database, for the document with ID=23454, all the dates in the domain are equally possible. Nevertheless, the document 34563 was issued the crisp (exact) date 11/12/1204. The time for documents 12211 and 23455 are specified by a possibility distribution. The first one is also known as a fuzzy number whereas the second one is also known as a fuzzy interval, as explained in Section 3.1.

3.2.3. Ill-known time interval

An ill-known time interval denoted by $[X, Y]$ is a precise time interval whose boundaries are not precisely known.

Definition 55. Ill-known time interval Let $\mathcal{T}$ be the time domain, and X, Y two ill-known values in the time domain. An ill-known time interval is given by $[X, Y]$. The evaluation of the ill-known time interval is given by equations (3.19),(3.20). The set of all the ill-known time intervals will be noted by $\mathcal{I}_{PVP}$.

3.2.3.1. Open ill-known time intervals

Quite often, the user may want to specify time intervals with open boundaries in one or both endpoints. Consider an ill-known time interval $[X, Y]$. Then it is possible to distinguish between the following two types of open intervals:

Definition 56. Completely unknown time interval: Both starting and ending points are unknown, therefore the whole interval is unknown.

Constraint	Rp	Rn
$C \triangleq (<, T)$	$>$	$\leq$
$C \triangleq (\leq, T)$	$\geq$	$<$
$C \triangleq (>, T)$	$<$	$\geq$
$C \triangleq (\geq, T)$	$\leq$	$>$

Table 3.4: Relations for the $\mathrm{Open}(C)$ function. Depending on the relation $R \in \{\leq, <, >, \geq\}$ in the constraint C, the values for Rp and Rn are shown.

Definition 57. Semi-open time interval: Only one of the two ill-known endpoints for the time interval $[X, Y]$ is unknown. Example 18 and Figure 3.6 illustrates a left open time interval.

3.2.3.2. Representation of semi-open time intervals

As mentioned before, the problem resides in the representation of semi-open time intervals.

Because of the ill-known constraints C_1, C_2, respectively defined on X and Y, a function called *Open* should be defined in order to deal with a proper representation of these intervals.

Definition 58. Open(C)
Consider an ill-known value T, a binary relation $R \in \{\leq, <, >, \geq\}$ and the constraint $C \triangleq (R, T)$. The function $\mathrm{Open}(C) = (I_p(C), I_n(C))$ provides both the possibility and necessity measure for all the points in the open part of a semi-open ill-known time interval and are defined by:

$$I_p(C) = \Big(\sup_{r \in \mathcal{T}, r\,\mathrm{Rp}\ w} \pi_T(w) \Big) \tag{3.46}$$

$$I_n(C) = \Big(\inf_{r \in \mathcal{T}, r\,\mathrm{Rn}\ w} 1 - \pi_T(w) \Big) \tag{3.47}$$

Where the values for the binary relations Rp and Rn are shown in Table 3.4.

A special constant *until changed*, UC [73] is used in temporal databases to specify an open time interval in which the ending point of the time interval is $+\infty$. Analogously, a special constant *from the beginning* FB, can be defined. In this case, the starting point of the time interval is $-\infty$.

When dealing with ill-known time intervals, the constants FB and UC can be redefined by using the function Open with the following parameters:

$$\mathrm{FB} = \mathrm{Open}((\leq, Y)) \tag{3.48}$$

$$\mathrm{UC} = \mathrm{Open}((\geq, X)) \tag{3.49}$$

Where the constraints C_1 and C_2 are given in equations (3.31) and (3.32).

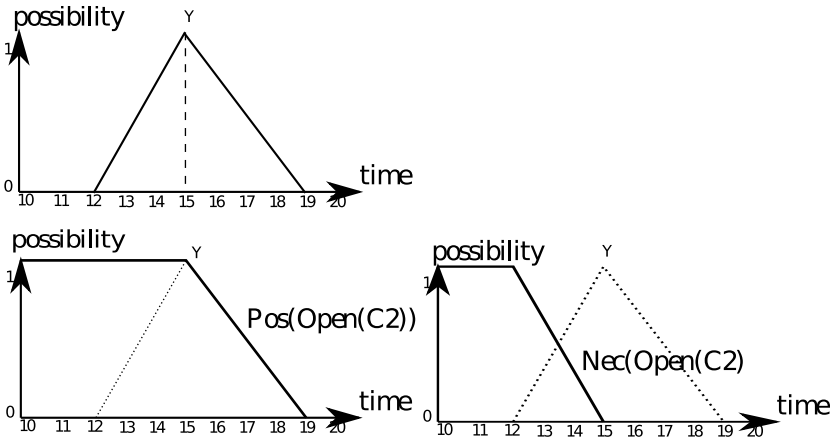


Figure 3.6: Possibility distribution for Y, and possibility and necessity measures for the open ill-known point, X

Example 18. Consider an ill-known time interval given by $[\mathrm{FB}, Y]$. Consider also that, in this case, $Y = [15/10/2012, 3, 4]$. Figure 3.6 shows the representation for Y. The user wants to obtain the possibility and the necessity measures for the FB part of the interval.

$$\mathrm{FB} = \mathrm{Open}(C_2) \text{ with } C_2 \triangleq (\leq, Y)$$
$$I_p(C_2) = \Big(\sup_{r \in \mathcal{T}_{\geq}\, w} \pi_Y(w) \Big)$$
$$I_n(C_2) = \Big(\inf_{r \in \mathcal{T}_{<}\, w} 1 - \pi_Y(w) \Big)$$

3.2.3.3. Data types

In order to properly represent an ill-known time interval in a database, some extra data types are needed. Because of the ill-known constraints, not all the combinations of data types for each ill-known time point (see Table 3.2) are allowed. Table 3.5 shows all the possible values that can be used to represent an ill-known time interval denoted by $[X, Y]$.

3.3. Crisp Valid-time Model for Relational DBs

This section is devoted to define a model for a crisp valid-time database. For the sake of simplicity, only the three main operations (CReate Update, Delete) in the Data Manipulation Language DML are shown. Usually the DML operations in a temporal database are re-defined (a typical update sentence in SQL could be expressed by means of a couple of insert and update sentences). Therefore, for the sake of clarity, high level primitives in the DML for a valid-time database are usually noted as Insert, Modify and

Subtype for X	Subtype for Y	Description
1 or 2	1 or 2	An ill-known time interval.
3	3	An unknown time interval.
FB	1 or 2	A left-open time interval.
1 or 2	UC	A right-open time interval.

Table 3.5: All the meaningful combinations of values for the time interval $[X, Y]$. The subtypes refer to Table 3.2.

Delete. In the following subsections each primitive is defined and explained. Finally, an illustrative example is given. For more complete information on the behaviour of a bi-temporal database, please refer to [183].

3.3.1. Temporal model for crisp databases

We will define the main concepts and elements in order to implement a model for temporal relational databases. First of all, we will introduce some definitions and notations.

Definition 59. Valid-time relation. Consider the following definitions and notations:

- A set of non-temporal attributes.

$$A = \{A_1, A_2, \ldots, A_n\} \tag{3.50}$$

  The domain for each attribute $A_1, \ldots, A_n$ is $D_1, \ldots, D_n$ respectively.

- The original primary key A_K is a subset of the attributes in A.

$$A_K \subseteq A \tag{3.51}$$

- Two attributes, S and E for the starting and ending points respectively. I defines the valid-time interval for the data.

$$I = (S, E) \tag{3.52}$$

  $\mathcal{T}$ is the time domain, and used as domain for S and E.

- Then R, the schema for the valid-time relation is:

$$R = A \cup I \tag{3.53}$$

- The primary key for the valid-time relation R is:

$$PK = A_K \cup I \tag{3.54}$$

- We will note by r any valid instance of R.[1] [2]

$$r \subseteq D_1 \times \ldots \times D_n \times \mathcal{T} \times \mathcal{T} \tag{3.55}$$

- $V(t)$ is the set of all the versions for a given tuple t. Formally,

$$V(t) = \{t_i \in r, t_i\,[A_K] = t\,[A_K]\} \tag{3.56}$$

  Obviously, t itself is included in this set.

We will illustrate the definitions with an example.

Example 19. Consider the set of attributes $A = \{A_1, A_2, A_3\}$. The primary key for these attributes is given by $A_K = \{A_1, A_2\}$. Let $I = \{S, E\}$ be the set of temporal attributes that define the validity period of the data. R is the valid-time relation and r is an instance of the relation. The instance r is given by the following elements. $r = \{(a_{11}, a_{12}, a_{13}, s_1, e_1), (a_{21}, a_{22}, a_{23}, s_2, e_2)\ , (a_{11}, a_{12}, a_{31}, s_3, e_3)\}$. The instance r is illustrated in Table 3.6. Consider the tuple $t_1 = (a_{11}, a_{12}, a_{13}, s_1, e_1)$. Then,

$$\begin{aligned} t_1\,[S] &= s_1 \\ t_1[E] &= e_1 \\ t_1[S, E] &= (s_1, e_1) \\ t_1\,[A_K] &= (a_{11}, a_{12}) \\ t_1\,[PK] &= (a_{11}, a_{12}, s_1, e_1) \\ V(t_1) &= \{t_1, t_3\} \end{aligned}$$

	$\mathbf{A}_1$	$\mathbf{A}_2$	A_3	S	E
t_1	a_{11}	a_{12}	a_{13}	s_1	e_1
t_2	a_{21}	a_{22}	a_{23}	s_2	e_2
t_3	a_{11}	a_{12}	a_{31}	s_3	e_3

Table 3.6: Example database containing the instance r of the valid-time relation R.

3.3.2. Data manipulation language

In a crisp temporal database, the Data Manipulation Language (DML) is re-defined in order to provide a consistent access to data and keep databases consistent. In the following we will implement the algorithms for the DML operations: Insert, Modify and Delete.

In order to simplify the algorithms for the manipulation of data, some auxiliary functions and constants are defined:

[1] Remark: Some authors, call r as the extension of the relation R. Here we will note t as a tuple of the instance r, but some authors will call t as the instance of the extension of the relation r. In the rest of the text we will follow the notation presented here.

[2] Note also that r is a set. Usually, in mathematical notation, a set is noted with capital letters, but here we will note with lower case letter to follow the relational notation.

Definition 60. From the beginning (FB).
Consider the elements in definition 59 and a tuple t. We will say $t[S] = FB$ when $t[S] = -\infty$.

Definition 61. Until changed (UC).
Consider the elements in definition 59 and a tuple t. We will say $t[E] = UC$ when $t[E] = +\infty$.

Definition 62. Current.
Consider the elements in definition 59. We will say that the tuple t is current in the instance r of the relation R when $t[E] = UC$.

For example, let us consider the last row in Table 3.7. The value for the time interval is $I =$ (S, E) $=$ (4/4/2012 , UC). The meaning is that the document with ID = 3 was valid the 4/4/2012 and it is still valid. The document with ID = 3 is current in the relation.

Example 20. Consider a historical database containing diplomatic documents. The starting and the ending dates denote when the diplomatic document is valid. It is possible that a diplomatic document is valid for a period of time and several years later it becomes valid again. The following elements are stored: an identifier of the document (ID), the entity that issues the document and the dates when the document is valid. Table 3.7 illustrates the first version of the database, after three insertions. In this example, $A =$ (ID, Entity) and $I =$ (Start, End).

ID	Entity	Start	End
3	E.U.	15/3/2012	30/3/2012
4	N.A.T.O.	25/3/2012	4/4/2012
5	C.E.I.	18/3/2012	2/4/2012
3	E.U.	4/4/2012	UC

Table 3.7: Example of historical database

Definition 63. Current (r, a_k).
Consider the elements in definition 59. The function Current (r, a_k) returns the crisp time interval of a tuple t with primary key a_k as follows:

$$\text{Current}\,(r, a_k) = \begin{cases} t[S, E] & \text{if } \exists t \in r : t[E] = UC \text{ and } t[A_K] = a_k \\ \emptyset & \text{otherwise} \end{cases}$$

For example, the function Current(r, 3) returns the time interval (4/4/2012 , UC). Conversely, Current(r, (4,'N.A.T.O.')) returns the empty set.

The Allen relations between two time intervals are shown in Figure 2.2. The complete implementation of the relations using only the starting and the ending points is defined in [19].

We will use two of the Allen relations: Overlaps and During which will be defined as follows.

Definition 64. Overlaps.
Given two time intervals defined by the couples of values $i_1 = (s_1, e_1)$ and $i_2 = (s_2, e_2)$, it is said that i_1 overlaps i_2 if:

$$i_1 \text{ overlaps } i_2 = (((s_1 < s_2) \wedge (e_1 > e_2)) \vee (e_1 < e_2)) \tag{3.57}$$

Definition 65. During.
Given two time intervals defined by the couples of values $i_1 = (s_1, e_1)$ and $i_2 = (s_2, e_2)$, it is said that i_1 during i_2 if:

$$i_1 \text{ during } i_2 = (s_1 > s_2) \wedge (e_1 < e_2) \tag{3.58}$$

In the rest of the this thesis, and without losing generality, we will consider that the time granularity are days. Also, the dates will be given in the format *dd/mm/yyyy*.

Definition 66. CloseR(i_1, i_2):
Consider two crisp time intervals defined by the couples of values $i_1 = (s_1, e_1)$ and $i_2 = (s_2, e_2)$. The CloseR(i_1, i_2) function allows to close the right-open interval i_1 with respect to the first value s_2 in i_2:

$$\text{CloseR}\,(i_1, i_2) = \begin{cases} (s_1, s_2 - 1) & \text{if } e_1 = UC \text{ and } i_2 \text{ During } i_1 \\ i_1 & \text{otherwise} \end{cases} \tag{3.59}$$

For example, consider two time intervals, $i_1 =$ (4/4/2012, UC) and $i_2 =$ (24/4/2012, UC). The result of applying the CloseR(i_1, i_2) is $i_1 =$ (4/4/2012, 23/4/2012).

Now it is possible to close the current version of an entity by using (3.59) and (3.57). This functionality is required to add or update new information about an existing entity in the relation. In the following we will assume that a new version starts *after* the current one.

Definition 67. Close-current(r, t).
Consider the elements in definition 59. The function Close-current(r, t) closes any current version t_k of the entity given by t and adds the new version t. For the implementation t_{CUR} and t_{UP} variables are defined:

$$\begin{aligned} t_{CUR}\,[A_k] &= t[A_K] \\ t_{CUR}\,[S, E] &= \text{Current } (r, t[A_K]) \\ t_{UP}[A_k] &= t[A_k] \\ t_{UP}\,[S, E] &= \text{CloseR } (t_{CUR}\,[S, E]\,, t[S, E]) \end{aligned} \tag{3.60}$$

Then, t_{CUR} is the current version of the entity given by the tuple t, and t_{UP} is the updated version of the tuple t_{CUR}. In this updated version, the time interval given by i_{UP} is closed with respect to the tuple t.

$$\text{Close-current}\,(r,t) = \begin{cases} \{(r \setminus t_{CUR}) \cup \{t_{UP}\} \cup \{t\}\} & \text{if Current } (r, t[A_K]) \neq \emptyset \\ r, \text{ otherwise} \end{cases}$$

For example, consider the document with ID = 3 in Table 3.7. The current version of the document started on 4/4/2012, the ending date was not specified. Then, a new version of the same document was issued on 24/4/2012. Then, the Close-current function closes the old version of the document and adds a new version. First, the function CloseR is applied with i_1 = (4/4/2012, UC) and i_2 = (24/4/2012, UC). Hence, the value for the time interval i_1 is (4/4/2012, 23/4/2012). Then, the modifications on the value for the time interval i_1 are stored. Finally a new row with the current values of the document and the time interval i_2 is stored. The result of this operation is illustrated on Table 3.8.

3.3.2.1. Modify

This operation adds new information about an existing entity (given by the tuple t) to the instance r of the relation R. The modify operation does not remove any previous value of the entity. It closes the current version and adds a new version.

Definition 68. Modify(r, t).
Consider the elements in definition 59. The algorithm for the modify operation is defined as follows.

$$\textbf{modify}\,(r,t) = \text{ Close-current } (r,t) \tag{3.61}$$

3.3.2.2. Insert

The user wants to store an entity (given by the tuple t) which is valid in the instance r of the relation R during the time interval specified by $i = (s, e)$. There are two main cases when performing a create operation:

1. The entity was never in the relation: The entity is added with the valid-time indicated by the crisp interval i.

2. The entity is in the relation. Depending on the value of the time interval, there are three possibilities:

   a) If the time interval i does not overlap with any other valid-time interval in the instance r relation R for the entity, then insert t in the instance r of the relation R. For example, consider that the document with ID = 3 was valid from 15/3/2013 to 30/03/2012. On 4/4/2012, a new version was issued. This version is still currently valid as illustrated in Table 3.7.

   b) If the ending point of the time interval for the existing entity in the database is *until changed*, then modify and close the current version of t and insert the new version. For example, consider now that the document with ID = 3 was valid on 24/4/2012. Here the problem is that the document with ID = 3

ID	Entity	Start	End
3	E.U.	15/3/2012	30/3/2012
4	N.A.T.O.	25/3/2012	4/4/2012
5	C.E.I.	18/3/2012	2/4/2012
3	E.U.	4/4/2012	23/4/2012
3	E.U.	24/4/2012	UC

Table 3.8: Example of historical database after insert operation.

was valid on 4/4/2012 but, for some reason, the ending date was not stored. If the document is again valid, then it is necessary to set the ending date and add a new row with the new starting date. This is illustrated in Table 3.8.

c) If the time interval i does overlap any existing valid-time interval for the entity t in the instance r of the relation R, then reject the insertion, For example, consider that the document manager wants to introduce a past valid-time for the document with ID = 3. The validity period for the document is $(6/4/2012, 25/4/2012)$. As the dates do overlap, it is not possible to reflect in a consistent way that the document was valid at that time interval. Therefore, the insertion is rejected.

Definition 69. Insert(r, t).
Consider the elements in definition 59. Then, the algorithm for the implementation of the insert operation is defined as follows.

$$\textbf{insert}\,(r,t) = \begin{cases} r & \text{if } \exists t_k \in V(t), (t[S,E] \text{ overlaps } t_k[S,E]) \\ r \cup \{t\} & \text{if } V(t) = \emptyset \text{ or } \forall t_k \in V(t), \\ & \neg\,(t[S,E] \text{ overlaps } t_k[S,E]) \\ \text{modify(r, t)} & \text{otherwise} \end{cases} \tag{3.62}$$

3.3.2.3. Delete

The delete operation logically removes a current entity t which is valid in the instance r of the relation R.

Definition 70. Delete(r, t).
Consider the elements in definition 59. The algorithm for the delete operation is defined as follows.

$$\textbf{delete}\,(r,t) = r \setminus V(t)$$

The set $V(t)$ is computed as explained in definition 59 and contains all the versions for the tuple t. In order to perform the deletion of a single version, the revise operation shoul be used.

For example, consider that the document manager wants to delete the history for the document with ID = 3. The result of this operation is shown in Table 3.9.

ID	Entity	Start	End
4	N.A.T.O.	25/3/2012	4/4/2012
5	N.A.T.O.	18/3/2012	2/4/2012

Table 3.9: Example of historical database after delete and revise operations.

3.3.2.4. Revise

This operation allows to make corrections in the values for an entity without affecting the validity period.

Definition 71. Revise(r, t).
Consider the elements in definition 59.

$$\textbf{revise}\,(r,t) = \begin{cases} \{(r \setminus \{t_k\}) \cup \{t\}\} & \text{if } \exists t_k \in V(t), t\,[S,E] = t_k\,[S,E] \\ r & \text{otherwise} \end{cases}$$

For example, consider now that the document manager wants to make a correction in the entity for the document with ID = 5. The new entity is 'N.A.T.O.'. The result of this operation is illustrated in Table 3.9.

3.3.3. Selection

The selection operator is very useful when the user wants to obtain a subset of tuples that fulfill a given constraint. More formally,

Definition 72. Selection operator.
Consider the elements in definition 59. The selection operator σ obtains a subset of tuples that fulfill the selection formula P from an instance r of the relation R. The selection formula usually consists on either a set of constraints or a logical expresion (for example, a Boolean condition). The selection operator is noted as follows:

$$\sigma_P\,(r) \tag{3.63}$$

Where $r \in R$ is the relation, and P is the selection formula. The selection formula is a set of two elements:

$$P = \left\{Q, Q^T\right\} \tag{3.64}$$

Where Q is the set of non-temporal constraints and Q^T the set of temporal constraints. The component Q is a set composed of atomic constraints:

$$Q = \{q_{\mathrm{a}_1}\theta \text{ val}_1, \ldots, q_{\mathrm{a}_n}\theta \text{ val}_n\} \tag{3.65}$$

Where:

- $q_{a \in A}$ is an atomic constraint. The constraint refers to an attribute a that belongs to the set of attributes $A \subseteq A_1 \ldots A_n$ of the relation R.
- θ is a relational operator; usually one of $\{=, \neq, <, \leq, >, \geq\}$.
- $val_1 \ldots val_n$ are values in the domain of the queried attribute.

The temporal constraints, Q^T are provided in a similar way. The main difference is that, instead of comparisons like $\{=, \neq, <, \leq, >, \geq\}$, we use the Allen relations [9]. Hence, the representation of the temporal constraints is expressed as follows:

$$Q^T = \{q_{v_1} \textbf{ AR } i_1, \ldots, q_{v_n} \textbf{ AR } i_n\} \tag{3.66}$$

Where

- $q_v, v \in I$ is an attribute representing a time interval. ($q_v = [s_v, e_v]$).
- **AR** is one of the thirteen Allen relations (equal, before, overlaps, starts, finishes, meets, during) and their respective reverses (See Figure 2.2).
- i_r is a crisp time interval with starting and ending points ($i_k = [s_k, e_k]$, $k = 1, \ldots, n$).

3.3.3.1. Query Evaluation

In a crisp relational database, the query satisfaction can be expressed by a Boolean value. The evaluation of the query requirements results in one of two possibilities; accept the record if it satisfies all the constraints or reject the record otherwise. The evaluation of the selection formula P given in equation (3.64) is handled as follows. For each tuple t in the database, two things happen independently:

- The crisp constraints expressed in Q are evaluated and aggregated. The result of this is a Boolean value. We will note as $e_Q(t)$ value for the evaluation of the constraints in Q for the tuple t.
- The temporal constraints expressed in Q are evaluated and aggregated. The result of this is again a Boolean value. We will note as $e_Q^T(t)$ the value for the evaluation of the constraints in Q^T for the tuple t.

The results from $e_Q(t)$ and $e_Q^T(t)$ are aggregated using a Boolean combination. For valid-time intervals, the preferred combination is the *'and'* ($\wedge$) operator. The function $e_{\text{final}}(t)$ is given by:

$$e_{\text{final}}(t) = e_Q(t) \wedge e_Q^T(t) \tag{3.67}$$

ID	Job	Works for	Start	Finish
1	Professor	4	5	10
2	Technician	4	3	7
3	Accountant	4	4	10
4	Administrator	-	1	UC
1	Professor	4	11	UC

Table 3.10: Employees table. Instance r of relation R.

ID	Adr.	Start	Finish
1	C/ Camino de ronda	FB	12
2	C/ Recogidas	FB	UC
3	C/ Pintor Maldonado	FB	UC
4	C/ Mesones	FB	UC
1	C/ Manuel de Falla	12	UC

Table 3.11: Address table. Instance s of relation S.

Example 21. Consider an employees database. The data are stored in two tables; Table 3.10 ($r \in R$) contains temporal data about the employees and Table 3.11 ($s \in S$) contains temporal data about the addresses for each employee. Consider now that the user wants to obtain all the employees who are a professor and that worked in the time interval $[7, 10]$.

The selection formula is the following:

$$\sigma_{\{\{r.Job='Professor'\},\{r.[S,E] \text{ Contains } [7,10]\}\}}(r) \tag{3.68}$$

The values for $e_Q(r)$ and $e_Q^T(r)$ for each record, as well as the final aggregation (see Equation (3.67)) are illustrated in Table 3.12. The resultset for the selection is shown in Table 3.13.

3.3.4. Cartesian Product

The relational Cartesian product is defined as all the possible combinations between tuples of r, s from two given relations R and S respectively. It is usually noted as $r \times s$. The temporal Cartesian product operator is defined as the relational Cartesian product with a predicate on the temporal valid-time intervals [184]. The temporal Cartesian product is defined then as $r \times^T s$.

ID	Job	Works for	Start	Finish	$e_Q(t)$	$e_Q(t)$	$e_{final}(t)$
1	Professor	4	5	10	True	True	True
2	Technician	4	3	7	False	False	False
3	Accountant	4	4	10	False	True	False
4	Administrator	-	1	UC	False	True	False
1	Professor	4	11	UC	True	False	False

Table 3.12: Intermediate calculations for selection formula given in equation (3.68)

ID	Job	Works for	Start	Finish
1	Prof.	4	5	10

Table 3.13: Resultset table for the selection in equation (3.68)

We will re-define the predicate given in [184] in terms of the Allen's relations. To do this, two auxiliary functions, intersect and overlapping interval should be defined.

Definition 73. Intersect(i_1, i_2)**.**
Given two intervals i_1 and i_2, the Intersect function returns a Boolean value showing whether the two intervals intersect or not. In terms of the Allen relations, two intervals i_1 and i_2 intersect if between them exist any of the Allen relations except Before or After. In other words, only if the Allen relations Before or After do not hold for the two intervals, they will intersect.

$$\text{Intersect}\,(i_1, i_2) = \{\neg\,(e_1 < s_2) \vee \neg\,(e_2 < s_1)\} \tag{3.69}$$

Example 22. Consider the following intervals $i_1 = [5, 10]$ and $i_2 = [0, 12]$. The intersect operation is calculated as follows:

$$\begin{aligned}\text{Intersect}\,(i_1, i_2) &= \{\neg\,(10 < 0) \vee \neg\,(12 < 5)\} \\ &= \text{True} \vee \text{True} \\ &= \text{True}\end{aligned} \tag{3.70}$$

The other auxiliary function that is required to define the predicate for the Cartesian product, needs some previous definitions as well.

Definition 74. Last(i_p, i_q)**.**
Given two time points i_p and i_q, this function obtains the lastest time point.

$$\text{Last}\,(i_p, i_q) = \begin{cases} i_p & \text{if } i_p \text{ After } i_q \\ i_q & \text{otherwise} \end{cases} \tag{3.71}$$

Note that because both i_p, i_q are time points, the After relation is implemented as $i_p > i_q$.

Definition 75. First(i_p, i_q)**.**
Given two time points i_p and i_q, this function obtains the earliest time point.

$$\text{First}\,(i_p, i_q) = \begin{cases} i_p & \text{if } i_p \text{ Before } i_q \\ i_q & \text{otherwise} \end{cases} \tag{3.72}$$

Note that because both i_p, i_q are time points, the Before relation is implemented as $i_p < i_q$.

Definition 76. Overlapping-interval(i_1, i_2)**.**
Given two intervals $i_1 = [s_1, e_1]$ and $i_2 = [s_2, e_2]$, this function returns an overlapping interval from the original two intervals.

$$\text{OvInt}\,(i_1, i_2) = \begin{cases} [\text{Last}\,(s_1, s_2)\,, \text{First}\,(e_1, e_2)] & \text{if Last}\,(s_1, s_2) \leq \text{First}\,(e_1, e_2) \\ \emptyset & \text{otherwise} \end{cases} \tag{3.73}$$

Now it is possible to define the temporal Cartesian product.

Definition 77. Temporal Cartesian Product.
Consider the elements in definition 59. The temporal Cartesian product of two temporal relations $r \in R = (A_1, \ldots A_n, S, E)$ and $s \in S = (B_1 \ldots, B_m, S, E)$ is noted as $r \times^T s$ and is defined by the following formula:

$$\begin{aligned} r \times^T s = \{z^{(n+m+2)} | \exists x \in r, \exists y \in s \\ \text{intersect}\left(x[S,E], y[S,E]\right) \wedge \\ z\left[A\right] = x\left[A\right] \wedge z\left[B\right] = y\left[B\right] \wedge \\ z\left[S,E\right] = \text{OvInt}\left(x\left[S,E\right], y\left[S,E\right]\right) \wedge z\left[S,E\right] \neq \emptyset\} \end{aligned} \tag{3.74}$$

Where A and B are a shorthand for $\{A_1, \ldots A_n\}$ and $\{B_1 \ldots, B_m\}$ respectively.

The following example illustrates the temporal Cartesian product.

Example 23. Consider the employee database mentioned before (Tables 3.10 and 3.11). Table 3.14 illustrates an intermediate step to compute the temporal Cartesian product.

3.3.4.1. Join

The join operator $\bowtie$ builds a new relation from two given relations, namely $r \in R$ and $s \in S$. This new relation is a set with all the possible combinations of tuples in both r and s that fulfill a predicate. It is usually noted as $r \bowtie_{a\theta b} s$ and called (theta) join, where a, b are attributes from r and s respectively and θ, is a relational operator. The temporal join definition is based on the temporal Cartesian product.

Definition 78. Temporal theta join.
Let r and s be two instances of relations R, S respectively. Then the temporal theta join is defined as follows:

$$r \bowtie^T_{a\theta b} s = \sigma_{a\theta b}\left(r \times^T s\right) \tag{3.75}$$

3.4. Possibilistic Valid-Time Model for Relational DBs

In this section we will formalize the model for possibilistic valid-time relational databases. The first subsection is devoted to the formalization of the model. Then, the data manipulation language is defined.

3.4.1. The generalized temporal model

The model is based on GEFRED [159] (Generalized Model of Fuzzy Relational DB) model. This model is extended by adding valid-time support which will be illustrated through definitions and examples. The information in the system is defined by the following elements:

r.id	s.id	$x\,[S,E]$	$y\,[S,E]$	[1]	[2]	[3]	$z\,[S,E]$
1	1	$[5,10]$	$[FB,11]$	True	11	5	$[5,11]$
1	2	$[5,10]$	$[FB,UC]$	True	UC	5	$[5,UC]$
1	3	$[5,10]$	$[FB,UC]$	True	UC	5	$[5,UC]$
1	4	$[5,10]$	$[FB,UC]$	True	UC	5	$[5,UC]$
1	1	$[5,10]$	$[12,UC]$	False	UC	12	-
2	1	$[3,7]$	$[FB,11]$	True	11	3	$[3,11]$
2	2	$[3,7]$	$[FB,UC]$	True	UC	3	$[3,UC]$
2	3	$[3,7]$	$[FB,UC]$	True	UC	3	$[3,UC]$
2	4	$[3,7]$	$[FB,UC]$	True	UC	3	$[3,UC]$
2	1	$[3,7]$	$[12,UC]$	False	UC	12	-
3	1	$[4,10]$	$[FB,11]$	True	11	4	$[4,11]$
3	2	$[4,10]$	$[FB,UC]$	True	UC	4	$[4,11]$
3	3	$[4,10]$	$[FB,UC]$	True	UC	4	$[4,11]$
3	4	$[4,10]$	$[FB,UC]$	True	UC	4	$[4,11]$
3	1	$[4,10]$	$[12,UC]$	False	12	12	-
4	1	$[1,UC]$	$[FB,11]$	True	UC	1	$[1,UC]$
4	2	$[1,UC]$	$[FB,UC]$	True	UC	1	$[1,UC]$
4	3	$[1,UC]$	$[FB,UC]$	True	UC	1	$[1,UC]$
4	4	$[1,UC]$	$[FB,UC]$	True	UC	1	$[1,UC]$
4	1	$[1,UC]$	$[12,UC]$	True	UC	12	$[12,UC]$
1	1	$[11,UC]$	$[FB,11]$	False	UC	11	-
1	2	$[11,UC]$	$[FB,UC]$	True	UC	11	$[11,UC]$
1	3	$[11,UC]$	$[FB,UC]$	True	UC	11	$[11,UC]$
1	4	$[11,UC]$	$[FB,UC]$	True	UC	11	$[11,UC]$
1	1	$[11,UC]$	$[12,UC]$	True	UC	12	$[12,UC]$

[1], intersect($x\,[S,E]\,, y\,[S,E]$)
[2], First($x[E], y[E]$)
[3], Last($x[S], y[S]$)

Table 3.14: Intermediate calculations for the temporal Cartesian product.

Definition 79. Generalized fuzzy domain.
Let D be the discourse domain, $\tilde{\mathcal{P}}(D)$ is the set of all possibility distributions defined on D, plus the NULL constant. The generalized fuzzy domain D_G is defined as:

$$D_G \subseteq \tilde{\mathcal{P}}(D) \cup \text{NULL} \tag{3.76}$$

The data types that can be used to represent D_G are shown in table 3.15.

Definition 80. Typeof(a).
Let D_G be a generalized fuzzy domain and consider the elements in definition 59. Let a be the value for the attribute A. The function typeof(a) returns the data type associated with the value a and returns a number in $[1, 10]$ as shown in Table 3.15.

$$\text{Typeof}(a) \mapsto [1, 10] \tag{3.77}$$

No.	Data type
1	A single scalar.
2	A single number.
3	A set of mutually exclusive possible scalar assignations.
4	A set of mutually exclusive possible numeric assignations.
5	A possibility distribution in a scalar domain.
6	A possibility distribution in a numeric domain.
7	A real number in $[0, 1]$ referring to degree of matching.
8	An *UNKNOWN* value.
9	An *UNDEFINED* value.
10	A *NULL* value.

Table 3.15: Data types

It is possible to define a more specific generalized temporal domain, $\mathcal{T}_G$.

Definition 81. Generalized fuzzy temporal domain.
Consider $\mathcal{T}$ to be the temporal domain, and let $\tilde{\mathcal{P}}(\mathcal{T})$ be the set of all *normalized* possibility distributions (see Section 3.1.1) defined on $\mathcal{T}$. The Generalized Fuzzy Temporal Domain, $\mathcal{T}_G$ is

$$\mathcal{T}_G \subseteq \left\{ \tilde{\mathcal{P}}(\mathcal{T}) \cup \text{NULL} \right\} \tag{3.78}$$

Note that $\mathcal{T}_G \subseteq D_G$. The data types for this domain have been studied previously in section 3.2 and are shown in tables 3.2 and 3.5.

A generalized fuzzy relation is defined in [159]. Here, we will extend the definition to a generalized fuzzy temporal relation.

Definition 82. Generalized fuzzy temporal relation.
Consider the elements in definition 59. Some of them will be extended for the fuzzy case.

- An attribute, V_{ID}, called version identifier will be added to the schema. This attribute is a counter for each different version of the entities.

- Then R_{FTG}, the schema for the fuzzy valid-time relation is:

$$R_{FTG} = A \cup V_{ID} \cup I \tag{3.79}$$

- The primary key for the fuzzy valid-time relation R_{FTG} is:

$$K_{GT} = A_K \cup V_{ID} \tag{3.80}$$

  A formal definition of the primary key for fuzzy valid-time relations will be given later in Definition 87.

- We will note by r any valid instance of R_{FTG}.

$$r \subseteq D_1 \times \ldots \times D_n \tag{3.81}$$

- $V(t)$ is the set of all the versions for a given tuple t. Formally,

$$V(t) = \{t_i \in r, t_i [A_K] = t [A_K]\} \tag{3.82}$$

  Obviously, t itself is included in this set.

- Let K_{GT} be the primary key for the valid-time relation as given in equation (3.80). Then, k denotes the values for the attributes in the primary key.

$$k = t [K_{GT}] \tag{3.83}$$

We will illustrate the definitions with an example.

Example 24. Consider the set of attributes $A = \{A_1, A_2, A_3\}$. The primary key for these attributes is given by $A_K = \{A_1, A_2\}$. Let $I = \{S, E\}$ be the set of temporal attributes that define the possibilistic validity period of the data. $R_{FTG} = A \cup V_{ID} \cup I$ is the valid-time relation and r is an instance of the relation. The instance r is given by the following elements. $r = \{(a_{11}, a_{12}, a_{13}, 001, s_1, e_1), (a_{21}, a_{22}, a_{23}, 001, s_2, e_2), (a_{11}, a_{12}, a_{31}, 002, s_3, e_3)\}$. The instance r is illustrated in Table 3.16. Consider the tuple $t = (a_{11}, a_{12}, a_{13}, s_1, e_1)$. Then,

$$\begin{aligned} s_1 &= t [S] \\ e_1 &= t[E] \\ i &= (s_1, e_1) \\ a_k &= t [A_K] = (a_{11}, a_{12}) \\ k &= t [K_{GT}] = (a_{11}, a_{12}, s_1, e_1) \\ V(t) &= r (t [A_K]) = \{t_1, t_3\} \end{aligned}$$

A generalized fuzzy temporal relation R_{FTG} can be noted also by:

$$R_{FTG} = (\mathcal{H}, \mathcal{B}) \tag{3.84}$$

Where $\mathcal{H}$ is the Head of the relation and consist on a fixed set of triplets (attribute, domain, compatibility) with an optional valid-time attribute:

	$\mathbf{A}_1$	$\mathbf{A}_2$	A_3	V_{ID}	S	E
t_1	a_{11}	a_{12}	a_{13}	001	s_1	e_1
t_2	a_{21}	a_{22}	a_{23}	001	s_2	e_2
t_3	a_{11}	a_{12}	a_{31}	002	s_3	e_3

Table 3.16: Sample database containing the instance r of the fuzzy valid-time relation R_{FTG}.

$$\mathcal{H} = \left\{ \left(A_{G1} : D_{G1} \left[, C_{A_{G1}}\right]\right), \ldots, \left(A_{Gn} : D_{Gn} \left[, C_{A_{Gn}}\right]\right), \left[\left(\text{PVP}, D_{\text{PVP}} \left[, C_{A_{\text{PVP}}}\right]\right) \right] \right\} \tag{3.85}$$

Note that D_{Gj} $(j = 1, \ldots, n)$ is the domain for the attribute A_{Gj}. $C_{A_{Gj}}$ is the compatibility attribute which takes values in the unit interval $[0, 1]$.

$\mathcal{B}$ is the body of the relation and it consists of a set of tuples. Each tuple is a triplet attribute- value- degree with an optional valid-time attribute:

$$\mathcal{B} = \left\{ \left(A_{G1} : \tilde{d}_{i1} \left[, c_{i1}\right]\right), \ldots, \left(A_{Gn} : \tilde{d}_{in} \left[, c_{in}\right]\right), \left[\left(\text{PVP}, \tilde{d}_{\text{PVP}} \left[, C_{A_{\text{PVP}}}\right]\right) \right] \right\} \tag{3.86}$$

The definition in [159] for R_{FTG} shows that classical relations are a particular case of this model.

Example 25. Consider a historical database containing diplomatic documents as explained in example 20. But now, the documents are from the Medieval Ages. Hence, the time is known with imprecision. For simplicity, the ill-known time points are represented as triangular fuzzy numbers. An ill-known time point is given by [dd/mm/yyyy, a, b]. The values for a and b are integers. Thus, the date given by [15/3/1012, 5, 2] is a triangular fuzzy number with the left bound the 10/3/1012, the core on the 15/3/1012 and the right bound on 17/3/1012.

$$\mathcal{H} = \left\{ \left(\text{ID} : D_{\text{ID}}\right), \left(\text{Entity} : D_{\text{Entity}}\right), \left(\text{PVP} : D_{\text{PVP}}\right) \right\} \tag{3.87}$$

Table 3.17 shows the elements Head, $\mathcal{H}$ and Body, $\mathcal{B}$ of the relation. When the compatibility degree is 1, it is omitted. The body, $\mathcal{B}$ consists on all the tuples shown in Table 3.17.

Definition 83. Value component.
The value component R^v_{FTG} of a fuzzy temporal relation R_{FTG} is a set with the value components for both the head and the body of the relation:

$$R^v_{FTG} = \{\mathcal{H}^v, \mathcal{B}^v\} \tag{3.88}$$

Where:

$$\mathcal{H}^v = \{A_{G1} : D_{G1}, \ldots, A_{Gn} : D_{Gn}\}$$
$$\mathcal{B}^v = \left\{ A_{G1} : \tilde{d}_{i1}, \ldots, A_{Gn} : \tilde{d}_{in} \right\}$$

$\mathcal{H}$	**ID**	Entity	Start	End
$\mathcal{B}$	3	E.U.	[15/3/1012, 5, 2]	[30/3/1012, 1, 1]
	4	N.A.T.O.	[25/3/1012, 3, 2]	[4/4/1012, 1, 7]
	5	C.E.I.	[18/3/1012, 4, 1]	[2/4/1012, 2, 2]

Table 3.17: Sample historical database

For example, in the case of the document with ID = 3:

$$\mathcal{H}^v = \{(\text{ID} : D_{\text{ID}}), (\text{Entity} : D_{\text{entity}}), (\text{PVP} : D_{\text{PVP}})\}$$
$$\mathcal{B}^v = \{3, \text{“E.U.”}, [15/3/1012, 30/3/1012]\}$$

Definition 84. Compatibility component.
The compatibility component R^c_{FTG} of a fuzzy temporal relation R_{FTG} is a set with the compatibility components for both the head and the body of the relation:

$$R^c_{FTG} = \{\mathcal{H}^c, \mathcal{B}^c\} \tag{3.89}$$
Where:
$$\mathcal{H}^c = \{[C_{A_{G1}}], \ldots, [C_{A_{Gn}}]\}$$
$$\mathcal{B}^c = \{[c_{i1}], \ldots, [c_{in}]\}$$

For example, in the case of the document with ID = 3:

$$\mathcal{H}^c = \{(C_{\text{ID}}), (C_{\text{entity}}), (C_{\text{PVP}})\}$$
$$\mathcal{B}^c = \{1, 1, 1\}$$

Definition 85. Temporal component.
The temporal component R^t_{FTG} of a fuzzy temporal relation R_{FTG} is a set with the temporal components for both the head and the body of the relation:

$$R^t_{FTG} = \left\{\mathcal{H}^t, \mathcal{B}^t\right\} \tag{3.90}$$
Where:
$$\mathcal{H}^t = \{(\text{PVP}, D_{\text{PVP}}[, C_{A_{\text{PVP}}}])\}$$
$$\mathcal{B}^t = \left\{\left[\text{PVP}, \tilde{d}_{\text{PVP}}[, C_{A_{\text{PVP}}}]\right]\right\}$$

For example, in the case of the document with ID = 3:

$$\mathcal{H}^t = \{(\text{PVP}, D_{\text{PVP}})\}$$
$$\mathcal{B}^t = \{15/3/1012, 30/3/1012\}$$

Analogously, it is possible to define both the name value component and the compatibility component for the temporal part.

Definition 86. Generalized primary key.
Consider D_G to be a fuzzy generalized domain, and let $A_{Gs} : D_{Gs}$ be the attributes

and the domain of the attribute for each $s \in S \subseteq \{1, \ldots, n\}$. A generalized primary key, K_G is a subset of the head:

$$K_G \subseteq \mathcal{H}, K_G = \{(A_{Gs} : D_{Gs})\}\, s \in S \subseteq \{1, \ldots, n\} \tag{3.91}$$

Subject to the following constraints:

$$\forall s \in S, \text{Typeof } (D_{Gs}) \in \{1, 2\} \tag{3.92}$$

$$\forall i, i' \in \{1, \ldots, m\}, \exists s \in S : (A_{Gs} : d_{is}) \neq (A_{Gs} : d_{i's}) \tag{3.93}$$

For example, consider the database in Table 3.17. Without any temporal constraint, the primary key K_G is:

$$K_G \subseteq \mathcal{H}, K_G = \{(\text{ID} : D_{\text{ID}})\}$$

In this case, the function *Typeof(ID) = 2* (see Definition 80 and Table 3.15). The primary key for the table is the attribute *ID*, a unique number. Two different documents have two different values for the *ID* attribute.

In order to add valid-time support, the primary key should be re-defined. E.g., consider the historical database. If the primary key is the *ID* attribute, a document should be valid only during one period of time. To resolve this problem, we extend the given primary key with a version identifier.

Definition 87. Generalized fuzzy temporal key.
Consider D_G to be a fuzzy generalized domain, and let $A_{Gs} : D_{Gs}$ be the attributes and the domain of the attribute for each $s \in S \subseteq \{1, \ldots, n\}$. Let V be a new attribute called *version*. A generalized fuzzy temporal key, K_{GT} is a subset of the head.

$$K_{GT} \subseteq \mathcal{H}, K_{GT} = \{(A_{Gs} : D_{Gs}) \cup (V_{ID} : D_{ID})\} \tag{3.94}$$

$$\text{Typeof } (D_{ID}) = s \in S \subseteq \{1, \ldots, n\} \tag{3.95}$$

Subject to the following constraints:

$$\forall s \in S, \text{Typeof } (D_{Gs}) \in \{1, 2\} \tag{3.96}$$

$$\forall i, i' \in \{1, \ldots, m\}, \exists s \in S : (A_{Gs} : d_{is}) \neq (A_{Gs} : d_{i's}) \tag{3.97}$$

For example, consider the database in Table 3.18. The primary key is now:

$$K_{GT} \subseteq \mathcal{H}, K_{GT} = \{(\text{ID} : D_{\text{ID}}), (V_{\text{ID}} : D_{\text{ID}})\}$$

3.4.2. Data manipulation language

The Generalized Fuzzy Relational Algebra [159] manipulates relations like R_{FTG}. The operations defined are: *Union, Intersection, Difference, Cartesian Product, Projection, Join* and *Selection*. Thus, in this section we will describe the following operations for temporal databases. The operations implemented are: *Insert, Modify* and

$\mathcal{H}$	**ID**	$\mathbf{V_{ID}}$	Entity	(Start	,End)
	3	001	E.U.	[15/3/1012, 5, 2]	[30/3/1012, 1, 1]
	4	001	N.A.T.O.	[25/3/1012, 3, 2]	[4/4/1012, 1, 7]
$\mathcal{B}$	5	001	C.E.I.	[18/3/1012, 4, 1]	[2/4/1012, 2, 2]
	3	002	E.U.	[4/4/1012, 3, 3]	UC

Table 3.18: Sample historical database

Delete. The semantics of the operations will be the same as those defined for a crisp temporal database, whereas the temporal representation is made by the possibilistic valid-time period and the ill-known constraints (see sections 3.1 and 3.2).

It is important to notice that while the result of the evaluation of any comparison between crisp time intervals is Boolean, the evaluation of any comparison between *PVPs* is a value in the unit interval. Therefore, for a crisp valid-time relation, it an overlapping among any valid-time interval is not possible. In other words, a crisp temporal database is always strictly consistent. Thus, from the point of view of a fuzzy temporal database, there are the following choices (see Figure 3.7):

Definition 88. Strictly consistent.
It is not possible that two versions of an entity are valid at the same time. Consider a tuple t in the instance r of the fuzzy valid-time relation R_{FTG} and $Vt = \{t_1, \ldots, t_n\}$ (see Definition 82). Then, a fuzzy temporal database is strictly consistent if:

$$\forall k, j \in \{1, \ldots, n\}, k < j : t_k \text{ Overlaps } t_j \leq 1 \tag{3.98}$$

Definition 89. Ill-consistent.
Two PVPs of the same entity do overlap. The following sub-types can be distinguished:

Definition 90. Co-existence
Two versions of the same entity may exist at the same time.

$$\forall k, \exists! j \in \{1, \ldots, n\}, k < j : t_k \text{ Overlaps } t_j \geq 1 \tag{3.99}$$

Definition 91. Weak-consistence
Several versions of the same entity may exist at the same time.

$$\forall k, \exists j \in \{1, \ldots, n\}, k < j : t_k \text{ Overlaps } t_j \geq 1 \tag{3.100}$$

In the implementation of the DML operations we will consider a strictly consistent temporal database. This is a natural extension of crisp temporal databases. This approach allows a definition for the DML operations which is similar to the ones defined in Section 3.3.

Since the time intervals are now possibilistic valid-time periods, PVPs, the auxiliary functions defined in equations (3.59) to (3.61) are the basis for the following auxiliary functions.

Definition 92. CloseR(i_1, i_2).
Consider the elements in Definition 82. The CloseR function closes the PVP given by

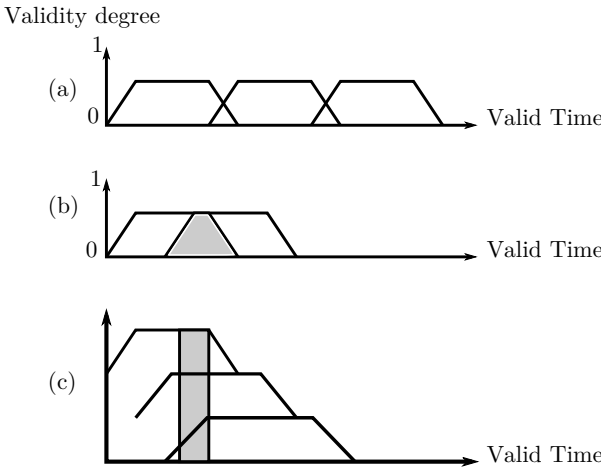


Figure 3.7: Classification for a fuzzy temporal database. The validity for an entity is represented by a trapezoid. Graphic (a) shows an example of a strictly consistent database. Between two versions of the entity some amount of overlapping is possible, but equation (3.98) always holds. Graphic (b) shows an example of a co-existence. Two versions of the same entity may overlap in any degree, but not more than two versions. Graphic (c) shows an example of weak-consistence. The different versions of the same entity are shown in an unfolded way, to clarify the multiple overlapping.

i_1 with a conjunctive combination of ill-known constraints (see section 3.1.4):

$$\text{CloseR}\,(i_1, i_2) = \begin{cases} i_1 = (s_1, e_z) & \text{if } i_1 = (s_1, UC) \\ i_1 & \text{otherwise} \end{cases} \tag{3.101}$$

Where $e_z \triangleq \{C_1\,(>, s_1)\,, C_2\,(<, s_2)\}$.

For example, consider i_1 = [[4/4/1012, 3, 3] , UC] and i_2 = [[15/4/1012, 2, 1], UC]. The result for CloseR(i_1, i_2) is i_1 = [[4/4/1012, 3, 3], [4/4/1012, 7/4/1012, 13/4/1012, 15/4/1012]]. The value that closes i_1 is a trapezoid in the form $[\alpha, \beta, \gamma, \delta]$ as defined in Section 3.1.3.

Definition 93. Close-current(r, t)**.**
Consider the elements in Definition 82. The function Close-current(r, t) closes any current version t_k of the entity given by t if it exists and add the new version t. In order to implement the functionality, the variables in equation (3.60) are used by equation (3.57).

$$\text{Close-current}\,(r, t) = \begin{cases} (r \setminus \{t_{CUR}\}) \cup \{t_{UP}\} \cup \{t\} & \text{if } t_{CUR} \neq \emptyset \\ r & \text{otherwise} \end{cases} \tag{3.102}$$

For example, consider the database in Table 3.18. The function Close-current (R_{FTG}, ID=3, [[15/4/1012,2,1], UC]) closes the current version of the patient with ID=3 and creates a new version with the specified time interval.

3.4.2.1. Modify

This operation adds new information about an existing entity (given by the tuple t) in the instance r of the fuzzy temporal relation R_{FTG}. The modify operation does not remove any previous value of the entity. Note that the modify operation is only applicable when the entity is current in the relation, i.e., $t \in r$ and $t[S, E] = (s, \mathrm{UC})$.

Definition 94. modify (r, t).
Consider the elements in Definition 82. The algorithm for the modify operation is defined as follows.

$$\textbf{modify}\,(r,t) = \text{ Close-current }\,(r,t) \tag{3.103}$$

3.4.2.2. Insert

The user wants to store an entity (given by the tuple t) which is valid in the instance r of the fuzzy temporal relation R_{FTG} during the time interval specified by the PVP, $i = (s, e)$. There are the following cases when performing an insert operation:

1. The entity was never in the relation: The entity is added with the valid-time indicated by the PVP, i. For example, consider the database given by Table 3.18. The following sentences correspond with the insertion of the first validity period for each document.

```
Insert(3,'E.U.',[[15/3/1012, 5, 2],
      [30/3/1012, 1, 1]]);
Insert(4,'N.A.T.O.',[[25/3/1012, 3, 2],
      [4/4/1012, 1, 7]]);
Insert(5,'C.E.I.',[[18/3/1012, 4, 1],
      [2/4/1012, 2, 2]]);
```

2. The entity is in the relation. Depending on the value of the time interval i, there are three possibilities:

   a) If the time interval i does not overlap any other valid-time interval in the relation R_{FTG}, then insert t in the instance r of the relation R_{FTG}. Note that here, the result of the Overlaps operator is a element of the unit interval. For example, the document with ID=3 is still valid. The insert sentence is the following.

   ```
   Insert(3,'E.U.',[[4/4/1012, 3, 3], UC]);
   ```

   b) If there exists a current version of the tuple, then modify and close the current version and insert a new version. For example, consider now that the document with ID=3 was valid around the 24/4/1012 (this is modelled by a triangular fuzzy number: $[24/4/1012, 1, 1]$). Here the problem is that the document with ID=3 was valid around 4/4/1012, but, for some reason, the ending date was not stored. If the document is again valid, then it is necessary to set the ending date and add a new row with the new starting date.

```
Insert(3,'E.U.', [[24/4/1012, 1, 1], UC]]);
```

c) If the time interval i does overlap with a degree of 1 any existing valid-time interval for the entity in the relation, then reject the insertion. For example, consider that the document manager wants to introduce a past validity period for the document with ID=3. The validity starting date for the document was around 6/4/12/1012 and the ending date was around 25/4/1012. As this interval does overlaps other time intervals with a degree of 1, it is not possible that the document was valid during two different time periods. Therefore, the insertion is rejected. The insert sentence is:

```
Insert(3,'E.U.',[[6/4/12/1012, 1, 1],
                        [25/4/1012, 1, 1]]);
```

Definition 95. insert (r, t).
Consider the elements in definition 82. The algorithm for the implementation of the insert operation is defined as follows.

$$\textbf{insert}\,(r,t) = \begin{cases} r \cup \{t\} & \text{if } t \notin r \text{ or } \forall t_k \in V\,(t)\,, (i \text{ Overlaps } i_k) < 1 \\ \text{modify(r, t)} & \text{otherwise} \end{cases} \tag{3.104}$$

Where $V(t)$ is the set of all the versions for the tuple t, as explained in Definition 82 and in equation (3.82).

3.4.2.3. Delete

The delete operation logically removes an entity which is valid in the instance r of the relation R_{FTG}.

Definition 96. delete (r, t).
Consider the elements in definition 82. The algorithm for the delete operation is defined as follows.

$$\textbf{delete}\,(r,t) = \{r \setminus V(t)\} \tag{3.105}$$

In order to delete a single version, the revise operator should be used. For example, consider that the document manager wants to delete the history for the document with ID = 3. The following sentence deletes all the rows for the documents with ID = 3.

```
Delete(3,'E.U.');
```

3.4.2.4. Revise

The revise operation allows the correction of non-temporal attributes for a given entity.

Definition 97. revise (r, t).
Consider the elements in definition 82. The algorithm for the revise operation is defined as follows.

$$\textbf{revise}\,(r,t) = \begin{cases} (r \setminus \{t_k\}) \cup \{\, t\} & \text{if } \exists t_k \in V(t), t\,[S,E] = t_k\,[S,E] \\ r & \text{otherwise} \end{cases} \tag{3.106}$$

For example, consider now that the document manager wants to make a correction in the organization for the document with ID = 5. The new organization is 'N.A.T.O.'. The required revise operation is:

```
Revise(5,'N.A.T.O.', [[18/3/1012, 4, 1],
                      [2/4/1012, 2, 2]]);
```

3.4.3. Selection

The selection operator is defined analogously as the crisp selection operator given in Definition 72 and the formal specification given by equation (3.63).

Definition 98. Fuzzy temporal relation.
Consider the elements in definition 82. The selection operator $\tilde{\sigma}^T$ returns a subset of tuples that fulfill a set of constraints P from an instance r of the relation R. The set of constraints is usually a Boolean combination of atomic constraints. The selection operator is noted as follows:

$$\tilde{\sigma}_P^{\mathrm{T}}\,(r) \tag{3.107}$$

Where $r \in R$ is the relation, and P is the selection formula.

In the possibilistic model, the main modification is that the selection predicate P is not a Boolean function. The selection predicate P returns a possibility distribution. The selection formula has the same appearance than the crisp selection formula in equation (3.64):

$$P = \left\{Q, Q^T\right\} \tag{3.108}$$

In this case, in order to evaluate both, where Q and Q^T could be fuzzy or crisp predicates, it is necessary to define the evaluation functions for both Q and Q^T. We will illustrate the possibilistic evaluation of Q. The possibilistic evaluation of Q^T is analogous. First of all, consider a Boolean function $\mathcal{B} : \mathbb{B}^n \to \mathbb{B}$. The evaluation function for the constraints in Q is given by:

$$\lambda : Q \to \mathbb{B} : \lambda(Q) \to \mathcal{B}\left(q_{a_1}\theta \text{ val}\,, \ldots, q_{a_n}\theta \text{ val}\,\right) \tag{3.109}$$

Now, the uncertainty about the evaluation of Q and a Boolean function $\mathcal{B} : \mathbb{B}^n \to \mathbb{B}$ is given by:

$$\pi_{\lambda(Q)} = \tilde{\mathcal{B}}\left(\pi_{q_{a_1}\theta \text{ val}}\,, \ldots, \pi_{q_{a_n}\theta \text{ val}}\right) \tag{3.110}$$

3.4.3.1. Query Evaluation

In fuzzy querying of regular (relational) databases, the modelling of query satisfaction is a matter of degree. Usually, the evaluation of the query requirements for a record results in a satisfaction degree s, where s lies in $[0, 1]$, where 0 denotes total dissatisfaction and 1 denotes complete satisfaction. In crisp querying, the evaluation of query requirements for a record results in accepting or rejecting the record as a part of the resultset. This can be modelled using satisfaction degrees, by assigning rejection a degree of 0 and acceptance a degree of 1 and not using any other value in $[0, 1]$.

The evaluation of the predicate $P = \left(Q, Q^T\right)$, is now handled as follows. For each tuple t in the database, with the valid-time notion of t being specified by a PVP j, two events happen independently:

- The preferences expressed in Q are evaluated, resulting in a satisfaction degree denoted here $e_Q(t)$. The presented model accepts any sound way of calculating this evaluation, as long as $e_Q(t) \in [0, 1]$.
- Depending on the Allen Relation selected **AR** , a specific set of ill-known constraints is considered. The possibility and necessity that t fulfills all these constraints are calculated using formulas based on equations (3.19) respectively (3.20) and aggregated using the $\min$ operator.

3.4.3.2. Aggregation and Ranking

In order to present the results to the user, a crude ranking method is used: for every tuple t, the sum of $\mathrm{Pos}_{Q^{time}}(t)$ and $\mathrm{Nec}_{Q^{time}}(t)$ gives an evaluation score $e'_{Q^{time}}(t)$ in interval $[0, 2]$. Because necessity cannot exceed 0 unless possibility is 1, this gives a natural ranking score. Some authors [77] mentioned before that the possibility and necessity measures result in a total order in the set of events. This $e'_{Q^{time}}(t)$ is then rescaled to the unit interval, resulting in $e_{Q^{time}}(t)$. The final ranking $e_{final}(t)$ is now given by a convex combination:

$$e_{final}(t) \;=\; \omega * e_Q(t) \;+\; (1 - \omega) * e_{Q^{time}}(t), \omega \in [0, 1] \tag{3.111}$$

The use of this convex combination allows a record to make up for a low score for the temporal constraint by a good score for the non-temporal constraint (or vice versa). Using different values for ω also allows granting the temporal constraint more weight with respect to the non-temporal constraint (or vice versa).

Example 26. Consider the example relation $c \in C$ given in Table 3.19 describing car models, containing general attributes (model name, manufacturer, car segment). The valid-time is referring to the approximate time period during which the car model was sold. In this example, the value for D is stored in *yyyy* format and a and b are represented by an integer. The ID field identifies a car model while the field Version ID (V_{ID}) identifies the instance for a car model, thus a car model in a certain state.

Consider the following query:

The user wants to obtain a list of models from segment B, sold by manufacturer Peugeot before the time period [2001,1,1], [2005,1,1].

ID	V_{ID}	Segment	Manufacturer	Name	Start	End
001	1	B	Peugeot	205	[1985,2,3]	[1997,2,1]
002	1	C	Peugeot	305	[1977,2,2]	[1989,2,3]
003	1	B	Citroen	C2	[2001,1,1]	[2005,1,1]
001	2	B	Peugeot	206	[2000,1,2]	[2011,2,1]
001	3	B	Peugeot	207	[2006,1,1]	[2011,1,1]

Table 3.19: Example database, instance $c \in C$

Using the introduced notations in (3.107),(3.108), the query is translated to:

$$\tilde{\sigma}^{\mathrm{T}}_{\{Q,Q^T\}}(c) \tag{3.112}$$

The query constraints are the following:

$$Q = (c.Segment = B) \wedge (c.Manufacturer = Peugeot) \tag{3.113}$$

$$Q^T = c.\,[S,E]\,\text{Before}\,[2001,1,1]\,,[2005,1,1] \tag{3.114}$$

ID	V_{ID}	Pos_{Q^T}	Nec_{Q^T}	$e_{Q^T}(rescaled)$	Q	e_{final} ($\omega = 0.5$)
001	1	1	1	1	1	1
002	1	1	1	1	0.5	0.75
003	1	1	0.5	0.75	0	0.375
001	2	1	0	0.5	1	0.75
001	3	0	0	0	1	0.5

Table 3.20: Result table and ranking

In the following we are going to detail the calculations for the tuple with ID = 003 and with V_{ID}=1. Q is the satisfaction of the non-temporal constraint. In this case since the user is looking for a Peugeot and the car with ID=003 is a Citroen, the result of the query is 0. The, the evaluation of the temporal part is as follows. First the possibility and necessity functions for *"Before"* $[2001,1,1]$, $[2005,1,1]$ are defined. The before operator is implemented by the ill-known constraint $C(<,S)$ where S is the starting point of the time interval, in this case $[2001,1,1]$. The possibility and the necessity distributions are given by a trapezoidal membership function, in the form $[\alpha,\beta\gamma,\delta]$. The possibility is given by [0, 0, 2001, 2002]. The necesity is given by [0, 0, 2000, 2001]. The possibility for the time interval of the car with ID = 3, given by [2001,1,1] is 1 and the necessity is 0.5. If we make the aggregation of both, we obtain 1.5 which has to be rescaled to the unit interval as 0.75. Then we are ready for the aggregation:

$$e_{final} = (0.5 * 0) + (0.5 * 0.75) = 0.375 \tag{3.115}$$

Where the value for ω is 0.5.

Table 3.20 shows a natural and gradual ranking for the results. The last record, (ID 001, V_{ID} 3) shows also that with ω = 0.5 both temporal and regular criteria have the same importance.

3.4.4. Cartesian Product

In this subsection we are going to extend the crisp version of the temporal Cartesian product to deal with ill-known time intervals. First of all, we need to define the ill-known counterpart functions of the previously defined *intersects, first, last, OvInt*.

Definition 99. Intersects(I,J).
Two ill-known time intervals intersect if one or both of the before and after relationships do not hold. Consider $I = [I_s, I_e]$ and $J = [J_s, J_e]$ be two ill-known time intervals with I_s, J_s the starting ill-known points and I_e, J_e the ending ill-known points of the intervals. Let the following ill-known constraints:

$$C_1 = (<, I_s) \tag{3.116}$$
$$C_2 = (=, J_s) \tag{3.117}$$
$$C_3 = (<, J_s) \tag{3.118}$$
$$C_4 = (=, I_s) \tag{3.119}$$
$$\begin{aligned}\text{Intersects}\,(I, J) &= \{\neg\,(C_1 \wedge C_2) \vee \neg\,(C_3 \wedge C_4)\} \\ &= \{\neg\,((C_1 \wedge C_2) \wedge (C_3 \wedge C_4))\}\end{aligned} \tag{3.120}$$

Definition 100. Last(I_n, J_m).
Given two ill-known points I_n and I_m, the last operator is a combination of the following ill-known constraints:

$$C_1 = (>, I_n) \tag{3.121}$$
$$C_2 = (=, J_m) \tag{3.122}$$
$$C_3 = (>, J_m) \tag{3.123}$$
$$C_4 = (=, I_n) \tag{3.124}$$
$$\text{Last}\,(I_n, J_m) = (C_1 \wedge C_2) \vee (C_3 \wedge C_4) \tag{3.125}$$

Analogously, the first operator is defined as follows.

Definition 101. First(I_n, J_m).
Given two ill-known points I_n and I_m, this operator is a combination of the following ill-known constraints:

$$C_1 = (<, I_n) \tag{3.126}$$
$$C_2 = (=, J_m) \tag{3.127}$$
$$C_3 = (<, J_m) \tag{3.128}$$
$$C_4 = (=, I_n) \tag{3.129}$$
$$\text{First}\,(I_n, J_m) = (C_1 \wedge C_2) \vee (C_3 \wedge C_4) \tag{3.130}$$

Now it is possible to define the function overlapping interval for the ill-known case.

Definition 102. Overlapping interval.
Given two ill-known time intervals I and J, this function returns an overlapping interval from the original two intervals. It is necessary to define two ill-known constraints:

$$C_1 = (\leq, \text{First}\,(I_e, J_e)) \tag{3.131}$$
$$C_2 = (=, \text{Last}\,(I_s, J_s)) \tag{3.132}$$

$$\text{OvInt}\,(I, J) = \begin{cases} [\text{Last}\,(I_s, J_s)\,, \text{First}\,(I_e, J_e)] & \text{if } C_1 \wedge C_2 \neq 0. \\ \emptyset & \text{otherwise} \end{cases} \tag{3.133}$$

Finally, the temporal Cartesian product for ill-known time intervals is defined as follows.

Definition 103. Temporal Cartesian product.
Consider the elements in definition 82. The temporal Cartesian product is notated $r \tilde{\times}^T s$ and is given by the following equation:

$$\begin{aligned} r \tilde{\times}^T s = \Big\{ z^{(n+m+2)} | \exists x \in r, \exists y \in s \\ \text{intersect}\,(x[S,E], y[S,E]) > 0 \wedge \\ z\,[A] = x\,[A] \wedge z\,[B] = y\,[B] \wedge \\ z\,[S,E] = \text{OvInt}\,(x\,[S,E]\,, y\,[S,E]) \wedge z\,[S,E] \neq \emptyset \} \end{aligned} \tag{3.134}$$

The temporal Cartesian product by ill-known constraints is illustrated in the following example.

Example 27. Consider a ill-known version of the employee's database, as illustrated by tables 3.21 and 3.22

ID	Job	Works for	Start	Finish
1	Prof.	4	$[5,1,1]$	$[10,1,1]$
2	Tech.	4	$[3,1,1]$	$[7,1,1]$

Table 3.21: Relation for the employees with an ill-known valid-time interval.

ID	Address	Start	Finish
1	C/ Camino de Ronda	$[1,1,1]$	$[12,1,1]$
2	C/ Recogidas	$[1,1,1]$	$[5,1,1]$

Table 3.22: Relation for the addresses with an ill-known valid-time interval.

3.4.4.1. Join

The possibilistic join operator $\tilde{\bowtie}$ builds a new fuzzy temporal relation from two given relations, namely $r \in R$ and $s \in S$. This new relation is a set with all the possible combinations of tuples in both r and s. It is usually noted as $r \tilde{\bowtie}_{a\theta b} s$ and called (theta) join, where a, b are attributes from r and s respectively and θ a relational operator. The possibilistic temporal join definition is based on the temporal Cartesian product.

r.id	s.id	intersect$(x[T], y[T])$	Last(I_s, J_s)	First(I_e, J_e)
1	1	1	$[5,1,1]$	$[10,1,1]$
1	2	1	$[5,1,1]$	$[5,1,1]$
2	1	1	$[3,1,1]$	$[7,1,1]$
2	2	1	$[3,1,1]$	$[5,1,1]$

Table 3.23: Intermediate calculations for the temporal Cartesian product.

Definition 104. Temporal theta join.
Let r and s be two instances of relations R, S respectively. Then the temporal theta join is defined as follows:

$$r \tilde{\bowtie}^T_{a\theta b} s = \sigma_{a\theta b}\left(r \tilde{\times}^T s\right) \tag{3.135}$$

3.5. Conclusions

In this chapter, we presented a complete valid-time model to represent and handle ill-known temporal intervals. The chapter includes the formal definition of possibilistic valid-time period in order to manage the time and the formal definition of ill-known constraints to define operators and integrity. This is the first formal model in the literature for possibilistic valid-time in relational databases. The semantics and the implementation of the DML operations are described within the chapter.

The novel contributions of this chapter are:

- A formal framework to model and handle imprecise time intervals, based on the possibility theory.
- A theoretical fuzzy temporal model for relational databases build on the top of the GEFRED model.

The next chapter is devoted to bipolarity in databases. First some preliminary concepts will be introduced. Then an approach to query in a bipolar way temporal databases will be explained.

4

Bipolar Querying of Temporal Databases

The contents of this chapter have been partially published on:

- T. Matthé, J. Nielandt, S. Zadrozny, and G. Tré, *Volume dedicated to prof. Patrick Bosc*, ch. Constraint-Wish and Satisfied-Dissatisfied: An Overview of Two Approaches for Dealing with Bipolar Querying.
- C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, *Eleventh International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets*, ch. Bipolarity in the Querying of Temporal Databases.
- C. Billiet, J. E. Pons, T. Matthé, G. De Tré, and O. Pons Capote, "Bipolar fuzzy querying of temporal databases," in *Lecture Notes in Artificial Intelligence*, vol. 7022, (Ghent, Belgium), pp. 60–71, Springer, Octobre 2011.
- C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, "Bipolar querying of Valid-time intervals subject to Uncertainty" in *Proceedings of the Flexible Querying Answering Systems conference*, Sep. 2013
- J. E. Pons, C. Billiet, O. Pons, G. de Tré, E. de Paermentier, and J. Deploige, "Consultas bipolares en bases de datos temporales: aplicación en bases de datos con datos históricos," in *Actas de las Jornadas Andaluzas de Informática 2011.*, pp. 150–156, 9 2011.

Contents

4.1. Introduction

Human beings usually express their preferences by using vague or fuzzy terms. For example, consider a car rental company that wants to buy *cheap* cars which have *low* fuel consumption. In order to translate these two 'fuzzy requirements' to a database query, fuzzy querying of regular databases has been proposed. A fuzzy query is composed by fuzzy terms which are interconnected by logical connectives. The research in this area include the modeling of linguistic terms (like *cheap* or *low*), the specification of query conditions by using fuzzy logic [185] and the study of the aggregation operators [50, 51, 175, 186].

The user preferences are modeled by using linguistic terms and / or soft aggregation operators. The tuples obtained as a result to a query are ranked to a degree, which shows to which extent the tuple satisfies the query conditions. Therefore, a *satisfaction degree* s is usually associated with each tuple in the result-set of the query. The satisfaction degree is usually in the unit interval $[0, 1]$. A value of 0 means that the tuple is not in the query result-set and the value 1 is given when the tuple fulfills all the query conditions. Intermediate values show partial query satisfaction. Sometimes, it is also possible that the user imposes a fulfillment threshold value δ in such a way that the tuples with satisfaction degree lower than δ are discarded from the result-set.

Usually, in fuzzy querying of databases, it is assumed that a tuple t which satisfies a query condition to a degree s, also satisfies its negation to a degree $s - 1$, in this sense, user preferences can be expressed in a more realistic way by using *bipolarity*. When dealing with preferences, bipolarity refers to the fact that users may express either positive and negative criteria or constraints and wishes. Positive criteria are used to express satisfactory, desired or acceptable values. On the other hand, negative constraints are used to express unsatisfactory, undesired or unacceptable values. In a similar way, constraints express what is accepted whereas wishes specify really desired values.

We will illustrate the differences between using positive / negative criteria with respect to constraints / wishes. For example, consider that the user is looking for a new car. A positive statement is 'I like a dark car', while 'I do not want a dark blue car' is a negative statement. It is possible to express similar preferences in term of constraints and wishes: 'I want a dark car' and 'if possible, I want a black car'. It is important to notice that often, negative conditions might be translated into constraints and positive conditions might be translated into wishes.

A specific situation in which the user specify both positive and negative conditions is the case when the user does not have a complete knowledge of the domain or if the the domain is too complex to specify.

Heterogeneous bipolarity holds when positive and negative criteria for a query condition are independent of each other. The modelling of queries by using this approach is closer to human reasoning.

As we have seen, bipolar querying provides more expression power than traditional fuzzy querying and in this chapter, two approaches to bipolar database querying are discussed. On the one hand, the *constraint-wish* (or mandatory-desired) approach will be presented, used amongst others by Dubois, Prade et al. [187, 188] and Bosc et al. [189–193], and on the other hand, the *satisfied-dissatisfied* (or positive-negative) ap-

proach will be discussed, used amongst others by Kacprzyk, Zadrożny et al. [194, 195] and De Tré, Matthé et al. [52, 196, 197]. For both approaches, an overview is given, which consecutively handles the semantics of the current framework, the evaluation of query conditions within this framework, the ranking of query results and the aggregation of compound query conditions.

The remainder of this chapter has been organised as follows: first, some preliminaries on bipolar query conditions will be presented in Section 4.2, explaining the two approaches that will be discussed in this chapter in more detail, together with their semantics. The next Section 4.3 discusses the ranking of the results of a bipolar query. Next, in Section 4.4, different techniques for aggregating the results of multiple query conditions are presented. The consideration or handling of temporal constraints in the bipolar criteria are studied in Section 4.5. In Section 4.6 a case of use is presented by using the medieval diplomatic database from the Low Countries.

Finally, Section 4.7 states some conclusions.

4.2. Bipolarity in the Query Conditions

The first work where a distinction between mandatory query conditions and desired query conditions has been done in [198]. As mentioned before, desired conditions specify what is considered to be positive whereas the opposite of a mandatory condition specifies what is considered to be negative.

There are two main approaches to use this idea with fuzzy querying techniques. On the one hand, the Twofold Fuzzy Set (TFS) based approach [187, 188] in which it is possible to specify a twofold fuzzy set as an elementary query condition with respect to a given attribute A. That twofold fuzzy set specifies which domain values are accepted and which domain values are really desired. This is known as the constraint-wish approach. On the other hand, the Atanassov (intuitionistic) Fuzzy Set (AFS) based approach allows the specification of desired and undesired domain values [196, 199]. This is known as the satisfied-dissatisfied approach.

Both approaches specify bipolarity *inside* elementary query conditions. In other words, the bipolar conditions are specified within the domain of a given attribute. When dealing with the evaluation of a condition, three (fuzzy) sets can be distinguished: positive, negative and not at all evaluated.

Nevertheless, it is possible to specify bipolarity *between* elementary query conditions. In this case, these conditions can be modeled by fuzzy sets, in particular it is possible to distinguish between mandatory and desired conditions [51, 185, 200–204]. Bipolar queries can be represented as a special case of the fuzzy 'winnow' operator [194]. An alternative proposal is to consider that a query is composed by a number of 'positive' and 'negative' elementary query conditions [52, 197]. The satisfaction and dissatisfaction of a query is computed then by the evaluation of positive and negative conditions.

In the following we will only consider bipolarity that is specified *inside* elementary query conditions. Two approaches are discussed in the following sections: the *constraint-wish* and the *satisfied-dissatisfied* approach.

4.2.1. Constraint-Wish Approach

A query in the *constraint-wish* approach in the universe of discourse U, has the following elements:

- A constraint C, identified by the membership function μ_C, which is the set of acceptable values of the universe U. Note that the rejected values can be computed as the complement of the fuzzy set C.
- A wish W, identified by the membership function μ_W, which is the set of desired values of the universe U.

It is important to notice that it is not coherent to wish something that is rejected, therefore a consistency condition can be imposed. There are two forms of consistency conditions:

1. *Strong consistency*: Once the element evaluated fulfills completely the constraint given by C, then the wish W is evaluated. In this case, the pair of fuzzy sets C and W then form a *twofold fuzzy set* [205].

$$\forall x \in U : \mu_C(x) < 1 \Rightarrow \mu_W(x) = 0. \tag{4.1}$$

2. *Weak consistency*: In this case, a relaxed condition is imposed on the wish W: a wish should be harder to satisfy than a constraint. In this case, the pair of fuzzy sets C and W then form an interval-valued fuzzy set (IVFS) [206–209].

$$\forall x \in U : \mu_W(x) \leq \mu_C(x). \tag{4.2}$$

In both cases, the bipolar query condition can be modeled by using an IVFS, since a twofold fuzzy set is a special case of it. An IVFS is given by:

$$F = \{(x, [\mu_{F_*}(x), \mu_{F^*}(x)]) | (x \in U) \wedge (0 \leq \mu_{F_*}(x) \leq \mu_{F^*}(x) \leq 1)\}. \tag{4.3}$$

Then, the bipolar query condition is modeled in the following way:

- The constraint is modeled by the upper membership function, $\mu_{F^*} = \mu_C$.
- The wish is modeled by the lower membership function, $\mu_{F_*} = \mu_W$. It can be seen that the wish has a secondary role in the query.

In natural language, a bipolar query condition in the constraint-wish approach can be read as '*satisfy C and, if possible, satisfy W*' [187]. The evaluation of an elementary bipolar query condition (in the form 'A is $F = (C, W)$') results in a pair of satisfaction degrees $(c(t), w(t)) \in [0, 1]^2$. Consider a tuple t and an attribute A, then $t[A]$ is the value of tuple t for attribute A:

$$c(t) = \mu_C(t[A]) \tag{4.4}$$

$$w(t) = \mu_W(t[A]) \tag{4.5}$$

4.2.2. Satisfied-Dissatisfied Approach

A query in the *satisfied-dissatisfied* approach in the universe U, has the following two elements which are independent one from the other:

- A set of values of an attribute A which are positively evaluated. The fuzzy set F^+ specifies the positive part of the condition.
- A set of values of an attribute A which are negatively evaluated. The fuzzy set F^- specifies the negative part of the condition.

These two sets can be modeled by *Atanassov (intuitionistic) Fuzzy Sets* (AFSs) [210]. More formally, an AFS F is defined as follows.

$$F = \{(x, \mu_F(x), \nu_F(x))|(x \in U) \wedge (0 \leq \mu_F(x) + \nu_F(x) \leq 1)\}. \tag{4.6}$$

with

$$\mu_F : U \rightarrow [0, 1] \tag{4.7}$$
$$\nu_F : U \rightarrow [0, 1] \tag{4.8}$$

In the following, we will adopt the notation μ_F, ν_F when referring to the sets F^+, respectively F^-. Where Eq. (4.7) is the membership function, and Eq. (4.8) is the non-membership function. A consistency condition is usually imposed on the AFS.

$$0 \leq \mu_F(x) + \nu_F(x) \leq 1, \forall x \in U \tag{4.9}$$

When dealing with a query, the consistency condition can be interpreted as one of the following equivalent forms:

1. For a given value x, the degree of non-preference $\nu_F(x)$ can never be larger than the complement of the degree of preference $(1 - \mu_F(x))$.
2. For a given value x, the degree of preference $\mu_F(x)$ can never be larger than the complement of the degree of non-preference $(1 - \nu_F(x))$.

Interval-valued fuzzy sets IVFSs and Atanassov (intuitionistic) Fuzzy Sets (AFSs) are equivalent from an operational point of view. Therefore both IVFSs and AFSs can be used in the constraint-wish approach. Nevertheless, the semantics of the AFSs are closer to the satisfied-dissatisfied approach. In the satisfied-dissatisfied approach, there is a total independence between positive and negative conditions. Hence, it is necessary to remove the consistency condition for the AFSs. A *bipolar AFS* is an AFS without the consistency condition. There are several proposals in the literature following these ideas; among them, the neutrosophic logic [211, 212], a fuzzy version of the Belnap's logic [213] proposed by Öztürk and Tsoukias [214] among others [215, 216]. It is important to notice that, in the satisfaction-dissatisfaction approach, the degrees of satisfaction and dissatisfaction are independent one from the other and they express the user preferences with respect to an attribute.

In natural language, a bipolar query condition in the satisfaction-dissatisfaction approach, can be read as '*preferably satisfy* F^+ *and preferably do not satisfy* F^-' [196]. The evaluation of an elementary bipolar query condition (in the form 'A is $F = (F^+, F^-)$') results in a pair of satisfaction degrees $\mu_F(x)$, $\nu_F(x)$. A bipolar satisfaction degree (BSD) is the couple $(\mu_F(x), \nu_F(x))$. The set of all possible BSDs is noted as $\tilde{\mathbb{B}}$. The following formula specifies the amount of information provided by a BSD [197]:

$$\mu_F(x) + \nu_F(x) \quad (\in [0, 2]). \tag{4.10}$$

The specification for the user preferences with respect to the value x might be classified according to Eq. (4.10) in three cases:

- *Full specification*: The user provides full and consistent information for his / her preferences. In this case, $\mu_F(x) + \nu_F(x) = 1$. Also, the consistency condition holds: $\nu_F(x) = 1 - \mu_F(x)$.
- *Underspecification*: The user provides less information for his / her preferences. In this case, $\mu_F(x) + \nu_F(x) < 1$ and the consistency condition does not hold. The lack of information is given by $1 - \mu_F(x) - \nu_F(x)$.
- *Overspecification*: In this case, the user provides inconsistent / conflicting information for his / her preferences ($\mu_F(x) + \nu_F(x) > 1$). The degree of conflict (the grade of contradiction [215]) is given by $\mu_F(x) + \nu_F(x) - 1$.

In the satisfaction-dissatisfaction approach, the evaluation of an elementary bipolar query condition is given by a pair of membership functions (μ, ν). A bipolar satisfaction degree BSD [196], is the pair $(s(t), d(t)) \in [0, 1]^2$ of values. Consider a tuple t and an attribute A, then $t[A]$ is the value of tuple t for attribute A:

$$s(t) = \mu_F(t[A]) \tag{4.11}$$

$$d(t) = \nu_F(t[A]) \tag{4.12}$$

4.2.3. Examples

In order to illustrate an elementary bipolar query condition, consider the case of a user who wants to buy a car. In this example we will only consider the engine size (in c.c.). In order to decide which engine size fits the user's needs, several criteria could be taken into account. For example, from the point of view of the power, the higher the engine size, the higher the power. On the other hand, with respect to the fuel consumption, the higher the engine size, the higher the fuel consumption.

Note that we are simplifying the problem; in this case, the engine power and the fuel consumption may have two separate bipolar scales. Nevertheless, both criteria will be evaluated and aggregated in positive and negative engine sizes.

When a user is looking for a car, a positive unipolar scale might be used to measure the power of the engine. Another negative unipolar scale could be used to measure the fuel consumption. We will consider the terms 'big' and 'too big' respectively for these two criteria. These terms describe the positive F^+ and negative F^- parts of the bipolar condition.

In the satisfied-dissatisfied framework, the preferences may be represented as illustrated in Figure 4.1(a). As a remark, an engine size of 2.700 c.c. is totally positively evaluated from the point of view of the engine power, and at the same time is totally negatively evaluated from the point of view of fuel consumption.

As mentioned in the previous section, it is usually possible to express in the constraint-wish approach what is expressed in the satisfied-dissatisfied approach. The positive condition is interpreted as the wish and the complement of the negative condition is interpreted as the constraint. In this example this is not possible, since the consistency condition is not met (See Eq. (4.1) and Eq. (4.2)). This is illustrated in Figure 4.1(b).

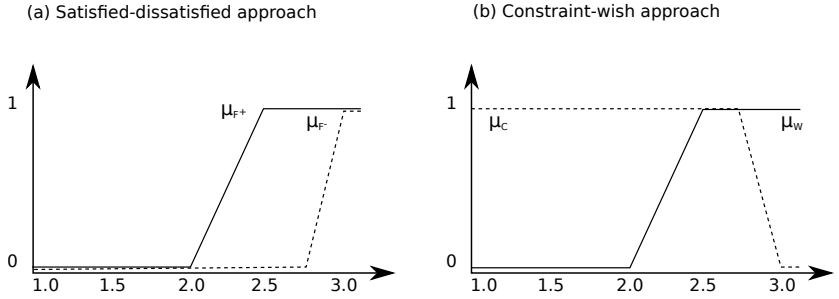


Figure 4.1: Examples: (a) μ_{F-}: 'too big', μ_{F+}: 'big'; (b) μ_C: 'not too big', μ_W: 'big'.

To illustrate the constraint-wish approach, we will consider a slightly different scenario. The user is looking for a car with a 'big' engine, but he or she would be happy with a car with an engine around 2500 - 3000 c.c. Then, the first condition could be interpreted as a constraint ('not big' engine is excluded). The second condition is just a desired size for the car engine. Figure 4.2(b) shows this example in the constraint-wish approach where the memberships functions are illustrated.

In this second example, the satisfied-dissatisfied approach is not suitable to represent those preferences. Figure 4.2(a) represents the following preferences: the user has positive feelings for a car with a engine of 2500 - 3000 c.c. and does not like small engines (in other words: not big engines).

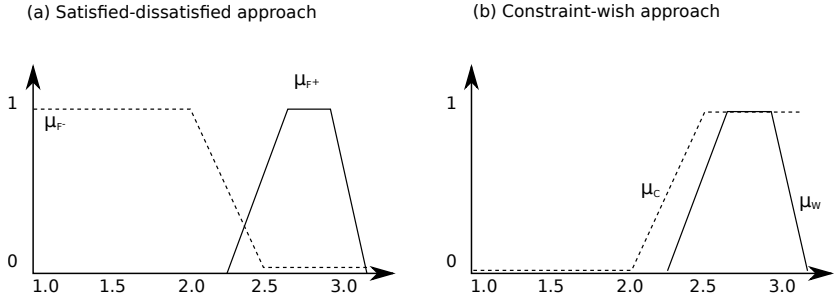


Figure 4.2: Examples: (a) μ_{F-}: 'not big', μ_{F+}: 'around 2500 - 3000 c.c.'; (b) μ_C: 'big', μ_W: 'around 2500 - 3000 c.c. '.

We are going to illustrate with an example, the weak consistency in the constraint-wish approach. Figure 4.3(b) shows the wish: 'preferred engine size is around 2500

- 3000 or slightly less'. The counterpart of this example in the satisfied-dissatisfied approach is shown in Figure 4.3(a).

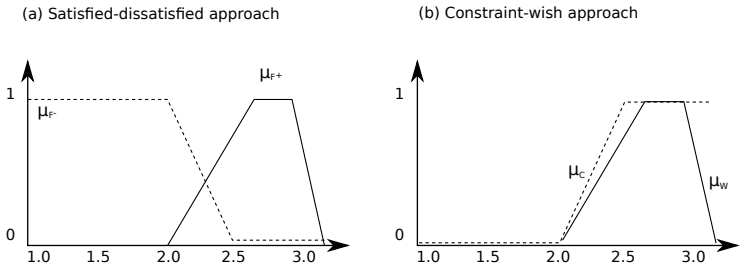


Figure 4.3: Examples: (a) μ_{F-}: μ_{F^-}: 'not big', μ_{F+}: '2500 - 3000 c.c. or slightly less'; (b) μ_C: 'not big', μ_W: 'around 2500 - 3000 c.c. or slightly less'.

4.3. Ranking of Query Results

As the result of the evaluation of a bipolar query condition, a pair of satisfaction degrees is obtained. For a given tuple t, the evaluation of a bipolar query condition in the constraint-wish framework results in a pair $(c(t), w(t))$ (See Eq. (4.4), (4.5)). In the satisfied-dissatisfied framework, the evaluation of a bipolar query condition results in BSD $(s(t), d(t))$ (See Eq. (4.11), (4.12)).

In this section we are going to study how to rank the query results in the constraint-wish framework as well as in the satisfied-dissatisfied framework.

4.3.1. Ranking in the Constraint-Wish Approach

In this approach, the ranking is usually done by using a lexicographical order with respect to the pairs (c, w). We will assume that constraints and wishes are not compensatory [189, 190] (a higher satisfaction of a wish would not compensate a lower satisfaction of the constraint). t_1 is preferred to t_2 if the following holds.

$$t_1 \succ t_2 \Leftrightarrow (c(t_1) > c(t_2)) \vee (c(t_1) = c(t_2) \wedge w(t_1) > w(t_2)) \tag{4.13}$$

There are some other proposals to rank query results in the constraint-wish approach. Some authors [194, 198], propose the application of a real function to each pair (c, w). This is known as scalarization. This same idea, but applied in a different context, has been studied before [217–219]. Recent research has been done in the interpretations of the 'and if possible' and its dual 'or at least' operators [220–222].

4.3.2. Ranking in the Satisfied-Dissatisfied Approach

In the satisfied-dissatisfied approach, both satisfaction and dissatisfaction degrees are considered to be independent. Therefore, when ranking query results, both degrees would have the same weight. In other words, both satisfaction and dissatisfaction degrees have the same impact on the ranking [196]. Intuitively, a symmetrical ranking would consider that the higher the satisfaction degree, the higher the ranking should

be, dually, the lower the dissatisfaction degree, the higher the ranking should be. This symmetrical ranking is formulated as follow.

$$r(s,d) = \frac{s + (1-d)}{2}. \tag{4.14}$$

The results of the Equation (4.14) are normalized in the unit interval ($r(s,d) \in [0,1]$). This ranking is parallel to one of the orders underlying Belnap's truth values bilattice [213]. Notice that this ranking gives the same weight to both degrees.

As a result of the evaluation, three special cases could be distinguished.

- *full satisfaction*: This is the case when $r(s,d) = 1$. Therefore, it must be that $s = 1$ and $d = 0$.

- *full dissatisfaction*: This is the case when $r(s,d) = 0$. Therefore, it must be that $s = 0$ and $d = 1$.

- *neutral*: This is the case when $r(s,d) = 0.5$. Therefore, it must be that $s = d$. In this case both BSDs $(0,0)$ and $(1,1)$ have a neutral ranking. In other words, these BSDs are situated in the middle of the ranking spectrum.

It is possible to define a different function for ranking a BSD. Nevertheless, in order to be consistent with the semantics of the BSDs, a ranking function r should meet the following requirements:

1. $0 \leq r(x,y) \leq 1$, with (x,y) a BSD, i.e., $r : \tilde{\mathbb{B}} \rightarrow [0,1]$.

2. $r(1,0) = 1$, The highest ranking is for the BSD with full satisfaction and no dissatisfaction.

3. $r(0,1) = 0$, The lowest ranking is for the BSD with full dissatisfaction and no satisfaction.

4. $\forall x, y \in [0,1] : r(x,x) = r(y,y)$, The ranking should be equal for values with equal satisfaction degree and dissatisfaction degree. The reason for this requirement is that, ranking wise, it is impossible to make a sensible distinction between the cases of total indifference (i.e., BSD $(0,0)$) and total conflict (i.e., BSD $(1,1)$), and also all other intermediate cases where $s = d$ (i.e., BSD (x,x), $x \in [0,1]$).

5. monotonicity: $r(x,y) \leq r(x+\epsilon, y)$ and $r(x,y) \geq r(x, y+\epsilon)$.

One of the main differences between the ranking in the constraint-wish approach and the ranking in the satisfied-dissatisfied approach, is that, it is not possible to use only one degree (either s or d) for ranking and the other degree as a 'tiebreaker'. Therefore, it is not possible to use the lexicographical order proposed for the ranking in the constraint-wish approach (see Eq. (4.13)), since the fourth requirement would not hold.

There are a plenty of ranking functions for the BSDs [223]:

$$r_1 = \frac{s + (1-d)}{2} \tag{4.15}$$

$$r_2 = \frac{s}{s+d} \tag{4.16}$$

$$r_3 = \frac{1-d}{(1-s)+(1-d)} \tag{4.17}$$

$$r_4 = \frac{s}{s+d} \cdot \frac{1-d}{(1-s)+(1-d)} \tag{4.18}$$

$$r_5 = \max\{0, s-d\} \tag{4.19}$$

$$r_6 = \min\{1+s-d, 1\}. \tag{4.20}$$

Equation (4.16) is discontinuous for the BSD $(0,0)$. Equation (4.17) is undefined for the BSD $(1,1)$ and finally, Equation (4.18) is undefined for BSDs $(0,0)$ and $(1,1)$. An extensive comparison on the behaviour and the properties of these functions is done in [223].

4.3.3. Comparison and discussion

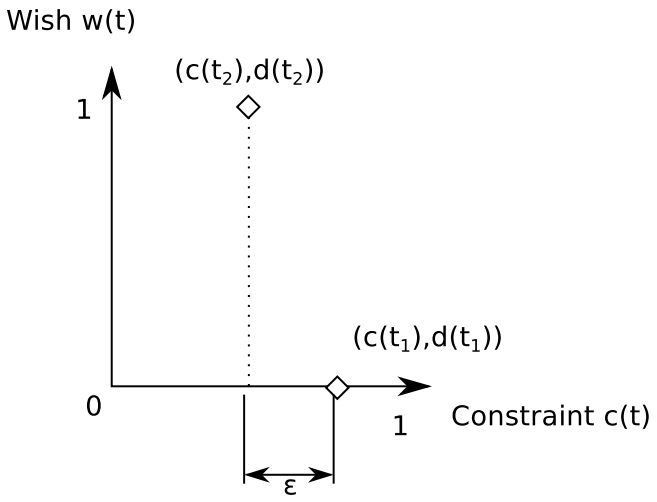


Figure 4.4: Example of the evaluation degrees for tuples t_1 and t_2. In this case, $c(t_1) = c(t_2) + \epsilon$, while $w(t_1) = 0$ and $w(t_2) = 1$.

The proposed ranking method (based on the lexicographical ordering) used in the constraint-wish approach makes wishes play a secondary role with respect to the constraints. This method would provide a counter-intuitive ranking in some cases. Figure 4.4 illustrates the ranking for two tuples t_1 and t_2, which have been evaluated by using a bipolar condition in the constraint-wish approach. The evaluation of the constraints and the wishes is given by the pairs $(c(t_1), w(t_1))$ and $(c(t_2), w(t_2))$ such that $c(t_1) = c(t_2) + \epsilon$, while $w(t_1) = 0$ and $w(t_2) = 1$. In this case tuple t_1 will be ranked before t_2, even for really small values of ϵ. One possible solution to this problem is to assume a discrete scale for c's and w's. This scale would have a small number of

levels. Then, it could be argued that the smallest difference in levels of the constraint satisfaction is large enough to justify its role in the ranking of the tuples, without taking into account the satisfaction degree for the wishes.

As explained in Section 4.3.2, the ranking in the satisfied-dissatisfied approach is based on the ranking of the BSDs. Both satisfaction and dissatisfaction degrees are considered to be mutually independent. Depending on the requirements of the application, the ranking function may be selected from a wide range of ranking functions. As a remark, note that a ranking function for BSDs have to fulfill the requirements specified in Section 4.3.2. We are going to describe some possible applications for the ranking functions r_1 to r_5 (See Equations (4.15) - (4.19)).

- The ranking function r_1 given by equation (4.15) is appropriate when the weights for $s(t)$ and $d(t)$ are equal. Query conditions are interpreted as preferences since tuples with $d(t) = 1$ or $s(t) = 0$ results in a neutral ranking.

- The ranking function r_2 given by equation (4.16) is appropriate when the positive condition is a requirement. In this case, $s(t) = 0$ has a ranking value of 0.

- The ranking function r_3 given by equation (4.17) is appropriate when the negative condition is a requirement. In this case, $d(t) = 1$ has a ranking value of 0.

- The ranking function r_4 given by equation (4.18) is appropriate when both positive and negative conditions are required.

- The ranking functions r_5 or r_6 given by equations (4.19), (4.20) are appropriate when the best of both conditions is needed. In this case, $r_5 = 0$ if $s(t) \leq d(t)$ and $r_6 = 1$ if $s(t) \geq d(t)$.

Summarising, the ranking in the constraint-wish approach is done by using the lexicographical order (Eq. (4.13)). It is important to notice that the situation presented at the beginning of this section (illustrated in Fig. 4.4) would not happen because of the consistency condition (See Eq. (4.1) for strong consistency and Eq. (4.2) for weak consistency). In the satisfied-dissatisfied approach, due to the independence between positive and negative conditions, the ranking might be done with different ranking functions, as presented in Eq. (4.15) -(4.20).

4.4. Aggregation of Bipolar Satisfaction Degrees

A bipolar query is usually composed of several elementary bipolar query conditions. In the constraint-wish approach, the evaluation an elementary bipolar query condition for a tuple t, results a pair $(c(t), w(t))$. Respectively, in the satisfaction-dissatisfaction framework, the evaluation of an elementary bipolar query condition results in a bipolar satisfaction degree, BSD $(s(t), d(t))$.

Therefore, for each elementary bipolar query condition, a pair of degrees is obtained. Then, all these pairs of degrees have to be *aggregated* to obtain a global result that reflects to which extent the tuple t satisfies the whole bipolar query.

For each framework, the aggregation is studied. In both approaches, we study the case when the pair of satisfaction degrees are treated individually and the cases when are treated as a whole.

4.4.1. Aggregation in the Constraint-Wish Approach

Given a bipolar query with n pairs of elementary bipolar conditions, the evaluation of these conditions for a tuple t gives a set of n pairs $(c_i(t), w_i(t))$ as a result. In order to obtain a global satisfaction degree, the set of n pairs have to be aggregated.

4.4.1.1. Treating $c(t)$ and $w(t)$ Individually

A bipolar query is composed by a list of n elementary query conditions. We will consider that the conjunction of all the n elementary query conditions is required [187, 188]. The constraints are aggregated in a conjunctive way; if a tuple t does not satisfy one of the constraints then it should be rejected. Nevertheless, the aggregation of wishes have different semantics. In this case, the wishes are aggregated in a disjunctive way; if a tuple fulfills a wish, then the tuple is desirable overall. More formally, the aggregation is given as follows.

$$(c(t), w(t)) = (\min_i c_i(t), \max_i w_i(t)). \tag{4.21}$$

Where the maximum and the minimum could be replaced by other aggregation operators (based on t-norms and t-co-norms). The problem of preserving the consistency is solved by treating the aggregation of wishes as the conjunction of the disjunction of wishes [188]:

$$(c(t), w(t)) = (\min_i c_i(t), \min(\max_i w_i(t), \min_i c_i(t))). \tag{4.22}$$

4.4.1.2. Treating $(c(t), w(t))$ as a Whole

This approach consider both the conjunction and the disjunction of elementary bipolar query conditions. Each pair $(c_i(t), w_i(t))$ is treated as a whole in the aggregation process [189, 190]. The conjunction of elementary bipolar query conditions is done by using a minimum operator, $lmin$. Respectively, the disjunction of elementary bipolar query conditions is done by using a maximum operator, $lmax$ [188–190]. A lexicographical order is assumed (see Subsection 4.3.1).

In order to illustrate this approach, we will consider two elementary bipolar queries in the form 'A_i IS F_i', $i = 1, 2$. The evaluation of these elementary bipolar queries results in two pairs of satisfaction degrees t: $(c_i(t), w_i(t))$, $i = 1, 2$. These pairs can be aggregated in a conjunctive or a disjunctive. The following equations describe the aggregations.

$$\begin{aligned}(c_{(A_1 \text{IS} F_1) \wedge (A_2 \text{IS} F_2)}(t), w_{(A_1 \text{IS} F_1) \wedge (A_2 \text{IS} F_2)}(t)) = \\ = lmin((c_1(t), w_1(t)), (c_2(t), w_2(t)) = \\ = \begin{cases} (c_1(t), w_1(t)) \text{ if } (c_1(t) < c_2(t)) \vee (c_1(t) = c_2(t) \wedge w_1(t) < w_2(t)) \\ (c_2(t), w_2(t)) \text{ otherwise} \end{cases}\end{aligned} \tag{4.23}$$

$$(c_{(A_1 \text{IS} F_1) \vee (A_2 \text{IS} F_2)}(t), w_{(A_1 \text{IS} F_1) \vee (A_2 \text{IS} F_2)}(t)) = \\ = lmax((c_1(t), w_1(t)), (c_2(t), w_2(t)) = \\ = \begin{cases} (c_1(t), w_1(t)) \text{ if } (c_1(t) > c_2(t) \vee (c_1(t) = c_2(t) \wedge w_1(t) > w_2(t)) \\ (c_2(t), w_2(t)) \text{ otherwise.} \end{cases} \quad (4.24)$$

The operators $lmin$ and $lmax$ (given in Eq. (4.23) and (4.24)) may be easily extended to the case of n elementary bipolar query conditions. The result of this aggregation is consistent in the sense of the equations (4.1) and (4.2).

4.4.2. Aggregation in the Satisfied-Dissatisfied Approach

A bipolar query in the satisfied-dissatisfied approach is composed by n elementary bipolar query conditions. Each tuple t is evaluated against the n elementary conditions. A set of n pairs of BSDs $\{(s_i(t), d_i(t)), i = 1, \ldots, n\}$ is obtained. These sets are aggregated in a global BSD $(s(t), d(t))$ which represents the global satisfaction and dissatisfaction with respect to all the elementary query conditions.

4.4.2.1. Treating s and d Individually

The aggregation of the BSDs is done by treating s and d individually [196, 197]. In this approach, conjunction, disjunction and negation between elementary query conditions can be computed. Also, it is possible to set weights to the query conditions. Therefore, the importance of the query conditions may be graded.

Elementary bipolar query conditions are modelled by bipolar AFSs (which are AFS without the consistency condition). The aggregation of BSDs is done in the same way AFSs are aggregated. In other words, the conjunction and disjunction are calculated following the semantics of the intersection and union of two AFSs. Öztürk and Tsoukiàs [214] proposed a continuous extension of Belnap's four-valued logic which coincides with these operations.

Non-Weighted Aggregation

In the following we will consider the conjunction and disjunction of two elementary bipolar query conditions in the form 'A_1 IS F_1' and 'A_2 IS F_2'.

Conjunction. The conjunction of two elementary bipolar query conditions in the form '$(A_1$ IS $F_1) \wedge (A_2$ IS $F_2)$' is computed by aggregating independently satisfaction and dissatisfaction degrees. Satisfaction degrees are aggregated by using the conjunction operator; two conditions are satisfied when both are satisfied. Dissatisfaction degrees are aggregated by using the disjunction operator; the conjunction of two conditions is dissatisfied when just one of them is dissatisfied. As mentioned before, the conjunction and disjunction operators are modeled by the min and max operators. Other operators based on triangular norms and co-norms can also be used. Hence, the conjunction of two elementary bipolar query conditions is computed by the following.

$$(s_{(A_1 \text{ IS } F_1)\wedge(A_2 \text{ IS } F_2)}(t), d_{(A_1 \text{ IS } F_1)\wedge(A_2 \text{ IS } F_2)}(t)) = \\ = (\min(s_{A_1 \text{ IS } F_1}(t), s_{A_2 \text{ IS } F_2}(t)), \max(d_{A_1 \text{ IS } F_1}(t), d_{A_2 \text{ IS } F_2}(t))). \quad (4.25)$$

Despite the fact that the aggregation presented for the constraint-wish approach given by Eq. (4.21) looks similar to the aggregation presented here in Eq. (4.25), they are semantically different. In Eq. (4.21), the minimum operator is applied to the constraint (in the satisfied-dissatisfied approach, the complement of the constraint is translated as the negative condition). Also, the maximum operator is applied to the wish (which is translated into the positive condition). AS we have seen, both aggregations Eq. (4.21) and Eq. (4.25) are completely different since they are applied to different query conditions.

Disjunction. The disjunction of two elementary bipolar query conditions in the form '$(A_1$ IS $F_1) \vee (A_2$ IS $F_2)$' is computed by aggregating independently the satisfaction and the dissatisfaction degrees. Analogous to the conjunction, the disjunction is done as follows. Satisfaction degrees are aggregated by using the minimum operator; the disjunction of two conditions is satisfied when just one of them is satisfied. Dissatisfaction degrees are aggregated by using the maximum operator; the disjunction of two conditions is dissatisfied when both are dissatisfied. Again, both min and max operator might be replaced by a triangular norm or co-norm, respectively. The conjunction is computed as follows.

$$(s_{(A_1 \text{ IS } F_1)\vee(A_2 \text{ IS } F_2)}(t), d_{(A_1 \text{ IS } F_1)\vee(A_2 \text{ IS } F_2)}(t)) = \\ = (\max(s_{A_1 \text{ IS } F_1}(t), s_{A_2 \text{ IS } F_2}(t)), \min(d_{A_1 \text{ IS } F_1}(t), d_{A_2 \text{ IS } F_2}(t))). \quad (4.26)$$

Negation. The negation of an elementary bipolar query condition ,'$\neg$ (A IS F)', is done by swapping fuzzy sets of the positive and the negative condition respectively. Therefore, '$\neg$ (A IS F)', is equivalent to 'A IS F'', where $F' = (F^-, F^+)$. The negation is computed as follows:

$$(s_{\neg(A \text{ IS } F)}(t), d_{\neg(A \text{ IS } F)}(t)) = (d_{A \text{ IS } F}(t), s_{A \text{ IS } F}(t)). \quad (4.27)$$

Weighted Aggregation

In order to grade the importance between different criteria, a weight for each criteria should be used. In the satisfied-dissatisfied approach, the weighted aggregation of the BSDs is defined [224]. Weights $w_i \in [0, 1]$ can be attached to each elementary query condition. A condition is fully important when $w_i = 1$. On the other hand, a condition is not important at all when $w_i = 0$. It is assumed that $\max_i w_i = 1$ [175].

When taking into account weights for the query conditions, the calculation of the impact of the weight for each BSD has to be done before the aggregation process starts. Next to that, the aggregation process is exactly the same as explained before. The weight influence on individual BSDs is calculated by the operator g:

$$g : [0, 1] \times \tilde{\mathbb{B}} \to \tilde{\mathbb{B}} : (w, (s, d)) \mapsto g(w, (s, d)). \quad (4.28)$$

Implication f_{im} and co-implication functions f_{im}^{co} are used to model the impact of weights. In this case both f_{im} and f_{im}^{co} return results in the unit interval $[0, 1]$. We will consider the Kleene-Dienes implication and co-implication:

$$\begin{aligned} f_{im_{KD}}(x, y) &= \max(1 - x, y) \\ f_{im_{KD}}^{co}(x, y) &= \min(1 - x, y). \end{aligned} \tag{4.29}$$

When dealing with the conjunction of BSDs, the impact of a weight is defined by the following.

$$g^{\wedge} : [0, 1] \times \tilde{\mathbb{B}} \to \tilde{\mathbb{B}} : (w, (s, d)) \mapsto g^{\wedge}(w, (s, d)) = \left(s_{g^{\wedge}(w,(s,d))}, d_{g^{\wedge}(w,(s,d))}\right) \tag{4.30}$$

where

$$\begin{aligned} s_{g^{\wedge}(w,(s,d))} &= f_{im}(w, s) \\ d_{g^{\wedge}(w,(s,d))} &= f_{im}^{co}(1 - w, d). \end{aligned}$$

If we use the Kleene-Dienes implication then we obtain:

$$g^{\wedge}(w, (s, d)) = (\max(1 - w, s), \min(w, d)) . \tag{4.31}$$

The conjunction operator $\wedge$ for BSDs is defined by

$$\wedge : (\tilde{\mathbb{B}})^2 \to \tilde{\mathbb{B}} : ((s_1, d_1), (s_2, d_2)) \mapsto (\min(s_1, s_2), \max(d_1, d_2)) \tag{4.32}$$

The definition of the weighted conjunction $\wedge^w$ of BSDs used the previously defined operator $g^{\wedge}$:

$$\begin{aligned} \wedge^w : ([0, 1] \times \tilde{\mathbb{B}})^2 &\to \tilde{\mathbb{B}} \\ ((w_1, (s_1, d_1)), (w_2, (s_2, d_2))) &\mapsto g^{\wedge}(w_1, (s_1, d_1)) \wedge g^{\wedge}(w_2, (s_1, d_1)). \end{aligned} \tag{4.33}$$

In the case of disjunction, the impact of a weight can be defined by the following:

$$g^{\vee} : [0, 1] \times \tilde{\mathbb{B}} \to \tilde{\mathbb{B}} : (w, (s, d)) \mapsto g^{\vee}(w, (s, d)) = \left(s_{g^{\vee}(w,(s,d))}, d_{g^{\vee}(w,(s,d))}\right) \tag{4.34}$$

where

$$\begin{aligned} s_{g^{\vee}(w,(s,d))} &= f_{im}^{co}(1 - w, s) \\ d_{g^{\vee}(w,(s,d))} &= f_{im}(w, d). \end{aligned}$$

Again, by using the Kleene-Dienes implication, the weight operator for disjunction is the following:

$$g^{\vee}(w, (s, d)) = (\min(w, s), \max(1 - w, d)) . \tag{4.35}$$

Let us consider the following as the disjunction operator $\vee$ for BSDs.

$$\vee : (\tilde{\mathbb{B}})^2 \to \tilde{\mathbb{B}} : ((s_1, d_1), (s_2, d_2)) \mapsto (\max(s_1, s_2), \min(d_1, d_2)) \tag{4.36}$$

Then, by using the definition of the weight impact operator $g^{\vee}$, the weighted disjunction is given by the following.

$$\begin{aligned} \vee^w : ([0, 1] \times \tilde{\mathbb{B}})^2 &\to \tilde{\mathbb{B}} \\ ((w_1, (s_1, d_1)), (w_2, (s_2, d_2))) &\mapsto g^{\vee}(w_1, (s_1, d_1)) \wedge g^{\vee}(w_2, (s_1, d_1)). \end{aligned} \tag{4.37}$$

Averaging

The aggregation of BSDs allows the usage of more advanced operators like, for example, the arithmetic mean (AM), geometric mean (GM) or harmonic mean (HM) [224]. An extended version of these averaging operators is necessary when dealing with the aggregation of BSDs. In the following we will illustrate the extension of the arithmetic mean (AM). A Similar extension can be defined for other averaging operators. The equation for the arithmetic mean is the following.

$$AM(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{4.38}$$

This formula cannot be applied directly to BSDs since BSDs consist of a pair of values. The satisfaction and dissatisfaction degrees have to be treated independently. Therefore, the extended version for the arithmetic mean is the following.

$$AM((s_1, d_1), \ldots, (s_n, d_n)) = \left(\frac{1}{n} \sum_{i=1}^{n} s_i, \frac{1}{n} \sum_{i=1}^{n} d_i \right). \tag{4.39}$$

Weighted Averaging

As mentioned before, a weight for each criteria can be used to grade the importance between them. A weight $w_i \in [0, 1]$ is linked with each condition, not with the satisfaction or dissatisfaction degrees of the condition itself. In order to obtain an appropriate scaling we will consider that $\max_i w_i = 1$. To compute the weighted averaging of BSDs, both satisfaction and dissatisfaction degrees are treated independently. In the following we will calculate the weighted mean (AM^w), but other operators like weighted geometric mean (GM^w), or weighted harmonic mean (HM^w)) can also be used.

$$\begin{aligned} AM^w &: ([0, 1] \times \tilde{\mathbb{B}})^n \to \tilde{\mathbb{B}} \\ ((w_1, (s_1, d_1)), \ldots, (w_n, (s_n, d_n))) &\mapsto \left(\frac{\sum_{i=1}^{n} w_i \cdot s_i}{\sum_{i=1}^{n} w_i}, \frac{\sum_{i=1}^{n} w_i \cdot d_i}{\sum_{i=1}^{n} w_i} \right). \end{aligned} \tag{4.40}$$

4.4.2.2. Treating (s, d) as a Whole

Ordered Weighted Averaging (OWA) operators for BSDs [224] are the basis for the aggregation BSDs treating (s, d) as a whole. A classical OWA [225, 226] operator has the following form (for n arguments $x_1, \ldots, x_n$):

$$OWA_W(x_1, \ldots, x_n) = \sum_{i=1}^{n} w_i \cdot x'_i \tag{4.41}$$

Where x'_i is the i^{th} largest value of $x_1, \ldots, x_n$ and $W = [w_1, \ldots, w_n]$. And $\sum_{i=1}^{n} w_i = 1$ are the vector weights. It is necessary to extend this operator to deal with BSDs. The first step is to rank the BSDs (see Section 4.3.2). There are several functions that perform the ranking of BSDs. The value (s'_i, d'_i) is the i^{th} largest BSD of $(s_1, d_1), \ldots, (s_n, d_n)$ according to the selected ranking function.

$$OWA_W : \mathbb{B}^n \to \mathbb{B} \tag{4.42}$$
$$((s_1, d_1), \ldots, (s_n, d_n)) \mapsto \left(\sum_{i=1}^{n} w_i \cdot s'_i, \sum_{i=1}^{n} w_i \cdot d'_i \right)$$

The selected weight vector changes the behaviour of the OWA operator. Note that the exact behaviour of the OWA operator depends also on the selected ranking function. The following three examples illustrate the different behaviour of the OWA operator by changing the weights:

- A maximum function, when $w_1 = 1$, $w_i = 0$ for $i > 1$.
- A minimum function, when $w_n = 1$, $w_i = 0$ for $i < n$.
- A median function, when
  - for odd n: $w_{\lceil \frac{n}{2} \rceil} = 1$, $w_i = 0$ for $i \neq \lceil \frac{n}{2} \rceil$, where $\lceil\ \rceil$ denotes the ceiling function.
  - for even n: $w_{\frac{n}{2}} = \frac{1}{2}$, $w_{\frac{n}{2}+1} = \frac{1}{2}$, $w_i = 0$ for $i \neq \frac{n}{2}$ and $i \neq \frac{n}{2} + 1$

4.4.3. Comparison and discussion

In this section we have presented several aggregation techniques for elementary bipolar query conditions. The aggregation process have different semantics between the constraint-wish approach and the satisfied-dissatisfied approach. Moreover, in order to provide a complete treatment of the aggregation of query conditions, we presented techniques that treat the pair of satisfaction degrees independently and techniques that treat the pair of satisfaction degrees as a whole.

The presented aggregations for the constraint-wish approach keep the semantics of constraints and wishes. The aggregation of $c(t)$ independently to $w(t)$, is done by considering a global constraint and a global wish whereas the aggregation of $(c(t), w(t))$ is done by using a lexicographical order.

The aggregation in the satisfied-dissatisfied approach might be more sophisticated due to the fact that both satisfaction and dissatisfaction degrees are independent one from the other. We have presented three possible aggregations operators when treating $s(t)$ and $d(t)$ independently.

- Aggregation based on the minimum triangular norm and the maximum triangular co-norm. This has been inspired by the aggregation of AFSs.
- Weighted aggregation, to grade the elementary bipolar query conditions.
- Averaging: The aggregation is performed by an averaging operator like the arithmetic, geometric or harmonic mean. In this case it is also possible to make a weighted averaging.

The selection of the aggregation method would depend on the application requirements, such as computation time, the need to better distinguish among the resulting tuples, etc.

Finally, we have presented the aggregation of BSDs as a whole. First the BSDs need to be ranked by using one of the presented methods. Next, an OWA operator is applied. As it has been shown, the aggregation depends on the specified weight vector. The selection of the weight vector depends on the application. From the point of view of the user, it is in general, less informative an aggregation method based on the ranking of BSDs. These type of aggregations methods, do not provide independent information relative to the satisfaction of both positive and negative query conditions. Nevertheless, the aggregation based on the OWA operators is useful when a quantifier aggregation is needed.

4.5. Bipolar querying of Temporal Databases

The bipolar querying of temporal databases consists on the application of bipolar techniques to the querying of temporal databases. In this section we will follow the satisfied-dissatisfied approach presented in Section 4.2.2. The specification of a query in the satisfied-dissatisfied approach is the following [52]:

$$\tilde{Q} = (Q^{pos}, Q^{neq}) \tag{4.43}$$

Where Q^{pos} represents the logical expression of positive query conditions and Q^{neg} represents the logical expression of negative query conditions. All the elementary query conditions in Q^{pos} and Q^{neg} can be specified in a 'fuzzy' way. Also, the elementary query conditions in both Q^{pos} and Q^{neg} can only be connected by a conjunction operator ($\wedge$) or a disjunction operator ($\vee$). Note that the negation operator is not allowed in the logical expressions due to the fact that it is considered to be non-involutive.

When departing from a query as specified in (4.43), a preference towards valid-time can be inserted at two levels:

- **Global level.** Here, the entire query is extended with a single time demand. This means that the entire query is given one elementary temporal constraint. The introduced temporal demand is defined by the user when the query is constructed and specifies a condition or constraint on the valid-time period of a tuple.
- **Local level.** Here, Q^{pos}, as well as Q^{neg}, are seen as logical aggregations of elementary query conditions and each of these elementary conditions is extended with a time demand. It is important to notice that the temporal constraint itself is neither positive nor negative. This means that every elementary condition is given an elementary temporal constraint which is evaluated and aggregated with the evaluation result of the condition. The introduced demands are defined by the user when the query is constructed and each one specifies a time constraint related with the fulfilment of the corresponding non-temporal condition.

In the following subsections, we will study the specification of time constraints at the global level first and then, the specification of temporal constraints at the local level.

4.5.1. Temporal constraints at the global level

In this approach we will consider valid-time temporal databases. The time is represented by possibilistic valid-time period (PVP) as described in Section 3.2.1. The query structure is detailed as follows. Then, the evaluation method is provided.

4.5.1.1. The Query Structure

The approach followed here introduces a global time demand: the user can define one time demand for the entire query. A query now has the following structure:

$$\left(Q^T, (Q^{pos}, Q^{neg})\right) \tag{4.44}$$

As in [52], Q^{pos} and Q^{neg} represent the positive and negative preference criteria, respectively, and Q^T represents the global temporal condition. Q^T is defined as follows.

$$Q^T = \{\ \mathbf{AR}\ q_t\} \tag{4.45}$$

Here, **AR** is one of the Allen relations and q_t is a possibilistic valid-time (*PVP*) specified by the user.

4.5.1.2. The Evaluation of the Query

In the presented approach, every tuple t from the instance r of the relation R contains a possibilistic valid-time period given by $t[S, E]$ to express the tuple's valid-time. The approach then is the following:

For every tuple t in the database, the bipolar query criteria Q^{pos} and Q^{neg} are evaluated, resulting in a BSD of the form (s_t, d_t), where s_t denotes the degree of satisfaction and d_t denotes the degree of dissatisfaction of the tuple. Separately, the temporal condition Q^T is evaluated in an attempt to define the degree to which t satisfies the query's temporal demand. For every tuple in the database, a validity satisfaction degree (*VSD*) is computed using the ill-known evaluation of the Allen relationships.

4.5.1.3. Presenting the Results to the User

This approach was designed to present the user both the resulting tuples and the degrees to which these tuples satisfy the entire query. To discover the tuples which are most useful to the user, the results have to be ranked. As the Q^T is evaluated independently from (Q^{pos}, Q^{neg}) for every tuple, the ranking for the VSD will be determined independently from the ranking for the BSD for every tuple. The BSDs are ranked as presented in Section 4.3.2. The VSDs are values in $[0,\ 1]$ and thus have a natural ranking.

To achieve the final ranking, the rank of the total BSD ($Rank_{BSD}$) and the VSD of a tuple are combined using a convex combination of both ranks:

$$Rank_{Total}\ =\ \omega * Rank_{BSD}\ +\ (1-\omega) * VSD$$

This somewhat unusual approach allows the overall ranking to reflect the effects of the BSD and the VSD in chosen proportions. Thus, by increasing the parameter ω, the non-temporal demands of the user can be given more importance and by lowering the ω, the time demand can be emphasized.

Example 28. Consider a car rental service which uses the valid-time temporal database given in Table 4.1. Next to some general attributes (Color, Fuel consumption (noted F.C., in l/100 km), Age (in years)), the relation shows the validity period of each car expressed by a PVP. The ID field uniquely identifies a physical car, but the properties of a car can be modified: some engine optimizations could allow less fuel consumption, changes in the color could be made, etc. E.g., the car described in tuples 1 and 5 is the same (both tuples have the same ID value), but their color differs. Therefore, a tuple in this database will be uniquely defined by the combination of the ID field and the Version ID (VID) field, which contains unique values for tuples with the same ID field value. A little remark: As the age of the car has nothing to do with when the car is available for rent, the 'Age'-attribute is considered as being user-defined time. Also, in this example, for the sake of simplicity, the chronons will be days.

					PVP	
ID	VID	Color	F.C.	Age	$[S,$	$E]$
001	1	Black	6	4	[10/12/2010, 10, 10]	[10/01/2011, 10, 10]
002	1	Red	5	6	[25/12/2010, 5, 5]	[25/02/2011, 5, 5]
003	1	Blue	4	2	[01/03/2011, 3, 3]	UC
004	1	Black	6	7	[01/01/2011, 10, 10]	[05/03/2011, 5, 5]
001	2	Red	6	4	[11/01/2011, 10, 10]	UC
002	2	Red	5	6	[26/02/2011, 10, 10]	UC
004	2	Black	6	7	[05/04/2011, 10,10]	UC

Table 4.1: The valid-time car relation. Note that the valid-time is represented by a PVP. The main point for both starting and ending points is a date in the dd/mm/yyyy format and the left and right margins are given in number of days.

ID	VID	$Rank_{BSD}$	VSD	$Rank_{Total}$		
				$\omega = 0$	$\omega = 0.5$	$\omega = 1$
001	1	-1	0	0	-0.5	-1
002	1	1	0.817	0.817	0.9085	1
003	1	0	0.142	0.142	0.071	0
004	1	-1	1	1	0	-1
001	2	1	1	1	1	1
002	2	1	0.338	0.338	0.669	1
004	2	-1	0	0	-0.5	-1

Table 4.2: Result set of the query

Consider the following query:

The user does not want to rent a black car, and prefers a red car which is either younger than 6 years or has an average fuel consumption that is around 5 or 6 litres/100 km and the car should be available around February 2011.

Using the structure (Q^{time}, (Q^{pos}, Q^{neg})) presented above, the query translates to:

$$(c^{time}, (c^{pos}_{Color} \wedge (c^{pos}_{Age} \vee c^{pos}_{Fuel}), c^{neg}_{Color}))$$

with

- c^{pos}_{Color} = $\{(Red,\ 1)\}$
- c^{neg}_{Color} = $\{(Black,\ 1)\}$
- c^{pos}_{Fuel} = $\{(5,\ 1),\ (6,\ 1),\ (7,\ 0.5)\}$
- c^{pos}_{Age} = $\{(0,\ 1),\ (1,\ 1),\ (2,\ 1),\ (3,\ 1),\ (4,\ 0.7),\ (5,\ 0.5),\ (6,\ 0.3)\}$

The temporal condition is represented by the following PVP.

$$c^T\ =\ [S, E]\ =\ [25/01/2011, 5, 5]\,, [10/03/2011, 5, 5]\,.$$

The resultset of the query, using the presented approach, is given in Table 4.2. Here, $Rank_{BSD}$ is the ranking of the BSD originating from the non-temporal bipolar constraints and $Rank_{Total}$ gives the total ranking (between -1 and 1) of the tuple with respect to the query, based on the chosen values for ω.

4.5.1.4. Discussion.

With $\omega\ =\ 0$, only the temporal constraint is taken into account. Because of the graduality of the VSDs, a richer ranking can be discerned than the simple acceptance or rejection which would occur when the query time interval would be requested to be fully contained in the tuple time intervals.

With growing values of ω, the non-temporal constraints grow in importance and the third tuple is ranked increasingly better than the fourth tuple, though the fourth tuple has a much higher VSD. This phenomenon can be desirable, as the fourth tuple has a much lower BSD rank.

With $\omega\ =\ 0.5$, only the tuple that scores high in both constraints gets a very high ranking (the fifth and the second tuples). Tuples with a low ranking for one of both constraints, are punished for this in the overall ranking, with the degree of punishment being relative to the lowness of the ranking. Vice versa, low scores for either the temporal or the non-temporal constraints can be compensated for with high scores for the other constraints. This introduces a very natural ordering, which ranks the tuples with respect to both the temporal and the non-temporal preferences of the user.

4.5.2. Temporal constraints at the local level

In this section, a novel technique is proposed to query a valid-time relation with valid-time indications subject to uncertainty in a heterogeneously bipolar way. The novelty of this technique is that it allows a local specification of the user's temporal preferences, i.e., for every elementary query condition, the user can specify during which crisp valid-time interval this condition should hold. First, the assumed structure of the valid-time relation is discussed, next the proposals for the construction of a query

ID	IID	DoB	AoO	VST	VET
1	1	11/08/77	Bruges	$[5/11/02, 1, 2]$	$[10/11/02, 1, 1]$
1	2	11/08/77	Ghent	$[16/11/02, 1, 1]$	$[24/11/02, 4, 3]$
1	3	11/08/77	Antwerp	$[10/06/03, 5, 3]$	$[11/08/04, 2, 2]$
2	1	23/05/77	Antwerp	$[16/11/02, 1, 1]$	$[24/11/02, 4, 2]$
2	2	23/05/77	Ghent	$[20/06/03, 4, 7]$	$[11/08/04, 4, 7]$

Table 4.3: A visualisation of an example relation, containing information on criminals. The value for attribute ID uniquely identifies a criminal, the combination of values for attributes ID and IID uniquely identify a state or version of a criminal.

and the evaluation of such a query are presented and last, some issues concerning tuple ranking and the technique adopted here are given.

We will consider valid-time relations with valid-time indications which are ill-known valid-time intervals, as described in Section 3.2.

4.5.2.1. Construction of the Query

In the current proposal, a query Q is constructed as:

$$Q = (c_{A_1}, tc_{A_1})\ op_1 \ldots\ op_{n-1}\ (c_{A_n}, tc_{A_n}).$$

Here, for every $i \in \mathbb{N}$, for which $1 \leq i \leq n$, every c_{A_i} is a bipolar elementary query condition specified using an AFS (as shown in Section 4.2.2) and every op_i is an operator, which can be either 'AND' or 'OR'. Every tc_{A_i} is now a temporal constraint. Such a temporal constraint tc_{A_i} is hereby any expression which can be expressed using a single Allen relationship and a single crisp time interval.

The interpretation here is that the user requires tuples for which the corresponding elementary query condition is valid during a time interval related to the time interval to which the temporal constraint evaluates. The nature of this relationship is given by the Allen relationship. Remembering that a tuple in a valid-time relation represents a state or version of a real object or concept (as opposed to the real object itself), the user's query demands are interpreted as demands towards the state of an object or concept during the given corresponding time period(s). This means that the user may express demands about the current state of an object or concept (by specifying a time interval containing the present) and / or demands about previous / future states of an object or concept (by specifying time intervals in history, respectively in the future). Thus, the user can describe the current, previous and / or future states of the object or concept he or she requires and thus some kind of required 'history'. Obviously, several elementary query conditions may concern the same attribute, but indicate a different time period.

Example 29. Let us consider a relation describing wanted criminals. This relation might contain the attributes 'Date of Birth (DoB)' and 'Area of Operation (AoO)', describing the date of birth and the main city of operation of a criminal. Table 4.3 shows an instance for the relation.

Considering the example relation given in Table 4.3, a user could be interested in identifying a criminal who 'was born somewhere in the summer of 1977, operated in the vicinity of Bruges from 6/11/02 until 10/11/02 and operated in the surroundings of

Ghent, but certainly not around Bruges any more, since 16/11/02'. The corresponding query would then be:

$$Q_{ex} = (c_{DoB},\ tc_{DoB})\ AND\ (c_{AoO,1},\ tc_{AoO,1})\ AND\ (c_{AoO,2},\ tc_{AoO,2})$$

where

- $(c_{DoB},\ tc_{DoB})$ models the criterion 'was born somewhere in the summer of 1977'.
  - Under the consideration that $\mathcal{T}$ is a time domain containing all dates in time,
    $$c_{DoB} = \{(x, \mu_{c_{DoB}}(x), 1 - \mu_{c_{DoB}}(x)) : x \in \mathcal{T}\}$$
    with membership function
    $$\mu_{c_{DoB}}(x) = \begin{cases} 0 & if\ x \text{ is a date with month in } \{Jan, Feb, Mar, Apr, Oct, Nov, Dec\} \\ 0.5 & if\ x \text{ is a date with month in } \{May, Sep\} \text{ and year = } 1977 \\ 1 & if\ x \text{ is a date with month in } \{Jun, Jul, Aug\} \text{ and year = } 1977 \end{cases}$$
    and the non-membership function $\nu_{c_{DoB}}$ is the inverse of the membership function $\mu_{c_{DoB}}$.
  - $tc_{DoB} = during\ ]-\infty, \infty[$, which reflects that there are no specific constraints on the valid-time for this criterion.
- $(c_{AoO,1},\ tc_{AoO,1})$ models the criterion 'operated in the vicinity of Bruges from 6/11/02 until 10/11/02'. Hereby
  - $c_{AoO,1} = \{(x, \mu_{c_{AoO,1}}(x), 1 - \mu_{c_{AoO,1}}(x)) : x \in Cities\}$ where the membership function $\mu_{c_{AoO,1}}$ is the one of the fuzzy set
    $$\{(Bruges, 1), (Ghent, 0.7), (Antwerp, 0.3)\}$$
    and the non-membership function $\nu_{c_{AoO,1}}$ is the inverse of the membership function $\mu_{c_{AoO,1}}$.
  - $tc_{AoO,1} = during\ [6/11/02, 10/11/02]$ models 'from 6/11/02 until 10/11/02'.
- $(c_{AoO,2},\ tc_{AoO,2})$ models the criterion 'certainly not around Bruges any more, since 16/11/02'. Hereby
  - $c_{AoO,2} = \{(x, \mu_{c_{AoO,2}}(x), 1 - \mu_{c_{AoO,2}}(x)) : x \in Cities\}$ where the membership function $\mu_{c_{AoO,2}}$ is the one of the fuzzy set
    $$\{(Bruges, 0.3), (Ghent, 1), (Antwerp, 0.7)\}$$
    and the non-membership function $\nu_{c_{AoO,2}}$ is the membership function of the fuzzy set
    $$\{(Bruges, 1), (Ghent, 0.3), (Antwerp, 0.3)\}.$$
  - Under the consideration that 'NOW' indicates the current date, $tc_{AoO,2} =$ *during* [16/11/02, NOW] models 'since 16/11/02'.

ID	IID	c_{DoB}	tc_{DoB}	$c_{AoO,1}$	$tc_{AoO,1}$	$c_{AoO,2}$	$tc_{AoO,2}$
1	1	(1,0)	(1,1)	(1,0)	(1, 0)	(0.3,1)	(0,0)
1	2	(1,0)	(1,1)	(0.7, 0.3)	(0, 0)	(1,0.7)	(1,0)
1	3	(1,0)	(1,1)	(0.3, 0.7)	(0, 0)	(0.7,0.3)	(1,1)
2	1	(0.5, 0.5)	(1,1)	(0.3, 0.7)	(0, 0)	(0.7,0.3)	(1,0)
2	2	(0.5, 0.5)	(1,1)	(0.7, 0.3)	(0, 0)	(1,0.7)	(1,1)

Table 4.4: The example query evaluation for the example table. The values for the non-temporal constraints show BSDs (s, d), the others pairs (p, n) consist of a possibility degree p and necessity degree n.

4.5.2.2. Query Evaluation

The query is evaluated for every tuple in the relation. For every tuple in the relation, the following happens distinctly:

- Every non-temporal elementary query condition is evaluated, resulting in a BSD for each. This evaluation is done as described in [197], using Eq. (4.11) and (4.12). The resulting BSD expresses the extend to which the tuple's value for the corresponding attribute satisfies and dissatisfies the user's non-temporal demand expressed in the elementary query condition.
- Every temporal constraint corresponding to an elementary query condition is evaluated, resulting in a possibility degree and a necessity degree. For this evaluation, the possibility and necessity degrees expressing respectively the possibility and necessity that the tuple's valid-time interval is in the given Allen relationship with the given crisp time interval are calculated using ill-known constraints. This calculation is done exactly as described in Chapter 3, and in [181], [227], using Eq. (3.19)-(3.20).

The interpretation here is that the resulting BSD expresses to which extend the state of an object or concept satisfies the user's request, while the possibility and necessity degrees express how plausible, respectively necessary it is that the object or concept under consideration is in this state during a time period related to the time period specified by the user.

In Table 4.4, the resulting BSDs, possibilities and necessities after evaluation of the individual criteria in the example query for the tuples of Table 4.3 are shown.

4.5.2.3. Object Ranking

The purpose of evaluating a query is of course to provide the user with the objects or concepts most fitting to his or her needs. In this case, two different criteria play a role.

1. The possibility and necessity degrees constitute quantifications of confidence in a context of valid-time uncertainty and thus portray the confidence in and necessity of the presence of an object able to fulfill the user's requests. These quantifications answer the question: 'How plausible is it that a suitable object or concept is available?'.

2. The (dis)satisfaction degrees constitute quantifications of satisfaction and dissatisfaction and thus portray the level of (dis)satisfaction an object could bring the user with respect to his or her demands. These quantifications answer the question: 'To what extend would a possibly available object (dis)satisfy the user's demands?'.

A fundamental question poses itself now: how can both quantifications be combined so as to obtain a single ranking of the results? An unambiguous and straightforward ranking allows to easily present the query results best fitting the user's demands. When ranking the results, the importance the user allocates to availability and (dis)satisfaction should be carefully examined and taken into account: some users might not care so much about availability, as long as they are sufficiently satisfied with the object, or vice versa. It is important to keep both quantifications as separate (meta)data in the ranked results presented to the users, else the mutually different interpretations of both quantifications would be lost.

In most existing proposals dealing with a similar situation, both quantifications are combined as to restrict each other. The result is generally seen as a quantification of the possibility that the user requirements are met. In the presented work, this same approach will be followed.

For every couple (c_{A_i}, tc_{A_i}) of a non-temporal elementary query constraint c_{A_i} and the corresponding temporal constraint tc_{A_i}, let

- $(s^t_{c_{A_i}}, d^t_{c_{A_i}})$ be the BSD and
- $(pos^t_{tc_{A_i}}, nec^t_{tc_{A_i}})$ be the possibility and necessity pair

all resulting from the evaluation of (c_{A_i}, tc_{A_i}) for a database tuple t. First, a score

$$sc^t_{c_{A_i}} = \frac{(s^t_{c_{A_i}} - d^t_{c_{A_i}} + 1)}{2}$$

is calculated, expressing how well the tuple fulfils the positive and negative non-temporal user demands about attribute A_i. This calculation is based on a scoring function suggested in [197] (but rescaled to cover the unit interval) and could be replaced by another consistent one. Now, the possibility $pos^t_{A_i}$ and the necessity $nec^t_{A_i}$ that the user's requirements about A_i are met, are calculated as follows:

$$\begin{aligned} pos^t_{A_i} &= \min(sc^t_{c_{A_i}}, pos^t_{tc_{A_i}}) \\ nec^t_{A_i} &= \begin{cases} 0 & \text{if } pos^t_{A_i} < 1 \\ \min(sc^t_{c_{A_i}}, nec^t_{tc_{A_i}}) & \text{else.} \end{cases} \end{aligned}$$

Every database tuple t represents an object or concept state. Let

$$\{t_{o,i} : i \in \mathbb{N} \wedge 1 \leq i \leq m\}$$

be the set of tuples $t_{o,i}$ representing states of object o. Then, for every such couple (c_A, tc_A), for every object or concept o, the degrees $pos^{t_{o,i}}_A$ and $nec^{t_{o,i}}_A$, $i \in \mathbb{N} \wedge 1 \leq i \leq m$ must be combined in a general possibility degree pos^o_A, respectively necessity degree nec^o_A, to express how possible (resp. necessary) it is that o meets the user's

ID	pos_{DoB}	nec_{DoB}	$pos_{AoO,1}$	$nec_{AoO,1}$	$pos_{AoO,2}$	$nec_{AoO,2}$
1	1	1	1	0	0.7	0
2	0	0	0	0	0.7	0

Table 4.5: The resulting possibility and necessity degrees.

demands about A. For this, a maximum function is used, to express that if any state of o has a high plausibility of meeting the user's demands, then o should be seen similar. Thus:

$$\begin{aligned} pos_A^o &= \max_{1\leq i\leq m}(pos_A^{t_{o,i}}) \\ nec_A^o &= \max_{1\leq i\leq m}(nec_A^{t_{o,i}}). \end{aligned}$$

The results of these calculations for the example are shown in Table 4.5. Based on these possibility and necessity degrees, a consistent ranking can be made. In the context of the presented work, it is suggested to model the 'AND' query operator with a minimum function and the 'OR' query operator with a maximum function. This would result in the objects with ID's respectively 1 and 2 having final possibilities 0.7, resp. 0 and final necessities 0, resp. 0.

4.5.2.4. Results and discussion

In this section, a novel technique is presented, to query a valid-time relation containing uncertain valid-time data in a heterogeneously bipolar way, allowing to assign a specific temporal constraint to every elementary query constraint. Furthermore, a major issue in combining quantifications of (dis)satisfaction with quantifications of confidence in a context of partial knowledge is described and shortly discussed. In the near future the possible interactions between valid-time uncertainty and bipolar querying will be further explored and some considerable attention will be dedicated to the issue in combining semantically different quantifications.

4.6. Case of use

In this case study, the medieval diplomatic database from the Low Countries (SMLC Diplomata Belgica) [228] provides data about diplomatic documents from the medieval ages. The temporal indications in this database are usually imprecise. In some cases the exact dates, the author or the place for a document are not exactly known. By using bipolar temporal queries, we may solve queries like:

"The historician wants to obtain all the documents written by the pope Alexander II and that were also written in a benedictine abbey and drawn up around the year 1703."

4.6.1. Medieval Diplomatic Sources of the Low-Countries in Belgium

The history department in the Ghent University and the Royal Commission of History in Belgium provide a database with diplomatic documents from the medieval ages (diplomas, letters and sources from the government). The main feature of this database is that most of the elements are known with some amount of imprecision, related, for instance, to the date of writing, the place or even the author. This database usually provides several fields that store the possible dates for an event.

The validity dates for a document have been studied by different historician researchers. Each document has a most possible data which is considered to be the most plausible date for that document. The database deals with valid-time for the documents and, therefore, the stored date is the date when the document was legally valid.

In this work, we will consider the most possible starting and ending dates provided by the historicians. The simplified schema for the database is shown in Table 4.6.

The description of the fields that we will use in the example are the following:

- ID: The document identifier. It is the primary key.
- Gender: The type of document. There are the following types: B a document from the pope, E a letter and D a document from the king.
- Language: The language in which the document is written. In the selected sample, L means that are written in latin.
- Author: The person or the authority that writes the document.
- Receiver: The person or the authority that receives the document.
- Valid time: The validity period for a document. It may be distinguished between the following two types:
  - The exact date: The date for a document is precisely known. In this case, the validity period is modelled by a rectangular possibility distribution. (An FVP in which $\alpha = \beta$).
  - The date is known with some imprecision. In this case, the validity period is represented by an FVP.

  Table 4.7 shows the validity periods associated to each element in Table 4.6.

4.6.2. Bipolar querying of temporal databases

We will explain how to extend the bipolar querying to temporal databases. In this case, we will work with valid-time databases. The query is specified as shown in Eq. (4.44) in Section 4.5.1.1 [20]. This allows to specify a validity period within a bipolar query. Therefore, the query $\tilde{Q}$ is specified as follows:

$$\tilde{Q} = (Q^{time}, (Q^{pos}, Q^{neg})) \tag{4.46}$$

ID	Gender	Languge	Author	Receiver
13559	B	L	Pope Innocentius III	UNKNOWN
13398	B	L	Pope Innocentius III	Marquis Boniface
13412	E	L	Count Hugues IV	King Henri I
13428	B	L	Pope Innocentius III	Marquis Boniface
13613	D	L	King Philippe	King Henri I
13790	B	L	Pope Innocentius III	King Kalojan
14268	D	L	King John	UNKNOWN

Table 4.6: Sample of the historical database from the medieval sources of the Low Countries.

ID	FVP	FVP in JDN format
13559	22/03/1203	[2160534, 2160534, -, -]
13398	20/05/1203	[2160593, 2160593, -, -]
13412	1/08/1203 +-20 days	[2160656, 2160676, -, -]
13428	7/02/1204 +- 30 days	[2160841, 2160871, -, -]
13613	12/11/1204	[2161135, 2161135, -, -]
13790	16/08/1205 +- 4 days	[2161410, 2161414, -, -]
14268	8/02/1209	[2162684, 2162684, -, -]

Table 4.7: Valid time in FVP representation.

As explained before, (Q^{pos}, Q^{neg}) refers to the positive and negative preferences for an user whereas Q^{time} specify the validity period for the result tuples. In the query, the time may be specified in an imprecise way, with respect to the starting and ending point of the interval. The time interval specification is interpreted in a conjunctive semantics. In other words, it is considered that the user prefers objects that are valid during the whole validity period.

4.6.3. Query evaluation

In the mentioned database, each row contains an interval V_r with the most possible starting s and ending e dates. For each row r in the database, both criteria (Q^{pos}, Q^{neg}) are evaluated independently. As a result, we obtain a BSD given by (s_r, d_r) with the satisfaction degree s_r and the dissatisfaction degree d_r. The temporal criteria Q_{time} is also evaluated independently. In this case, the evaluation of the temporal criteria is made with the gradual inclusion of the two sets:

$$\begin{aligned} deg_{vs}(\tilde{V}_q,\ \tilde{V}_r) &= deg(\tilde{V}_q \subseteq \tilde{V}_r) \\ &= \frac{card(\tilde{V}_q \cap \tilde{V}_r)}{card(\tilde{V}_q)} \end{aligned}$$

As the time intervals are continuous, the last formula may be written as follows:

$$deg_{vs}(\tilde{V}_q,\ \tilde{V}_r) = \frac{\int_U min(\mu_{\tilde{V}_q}(x),\ \mu_{\tilde{V}_r}(x))dx}{\int_{x \in U} \mu_{\tilde{V}_q}(x)dx}$$

Where U is the time domain and $\mu_{\tilde{V}_q}$ and $\mu_{\tilde{V}_r}$ are the membership functions for $\tilde{V}_q$ and $\tilde{V}_r$. Then, the Valid-time Satisfaction Degree (VSD) is obtained.

4.6.4. Aggregation

Once both, the bipolar satisfaction degree BSD and the Valid-time satisfaction degree are obtained, it is necessary to define an aggregation method. This method will specify the balance between the temporal and the bipolar criteria. Both values BSD and VSD are computed independently and are aggregated as shown in Section 4.5.1.3:

$$\text{Rank}_{\text{Total}} = \omega * \text{Rank}_{BSD} + (1 - \omega) * VSD \tag{4.47}$$

Example 30. Consider the database of diplomatic documents with the structure shown in Tables 4.6 and 4.7. The user wants to make the following query:

"To obtain the documents that have been written by the pope Inocentius III and that have been received by Marquis Boniface or have a gender equal to 'D'. The document was not received by Kalojan and it existed at least from the beginning of the year 1203 to the ending of the year 1206."

The query is expressed according to the Eq. (4.46) into the following:

$$\left(c^{time}, (c^{pos}_{Author} \wedge (c^{pos}_{Receiver} \vee c^{pos}_{Gender})), c^{neg}_{Receiver}\right) \tag{4.48}$$

Where:

- c^{time} (Expressed as FVP): [26/05/1203, 05/06/1203, 26/01/1206, 5/02/1206]. In Julian Day format (JDN): [2160599, 2160609, 2161275, 2161585].
- c^{pos}_{Author} = (Inocentius III,1).
- $c^{pos}_{Receiver}$ = (Marquis Boniface, 1).
- c^{pos}_{Gender} = (D, 1).
- $c^{neg}_{Receiver}$ = (Kalojan, 1).

Now, the evaluation of the query is done in two steps. First, the bipolar satisfaction degree BSD is computed. Then, the Valid-time satisfaction degree is computed. Table 4.8 shows the calculations. For each record, the satisfaction, dissatisfaction, BSD and VSD are shown.

Finally, the aggregation of the results is done by using Eq. (4.47). Table 4.9 shows how different values for ω give more weight to the temporal or the bipolar part of the query. The final ranking is a value in the interval $[-1, 1]$. This allows to classify the results in one of the three groups as explained in Section 4.3.2. A value of -1 is a total dissatisfaction. A value of 0 represents indifference. A value of 1 represents full satisfaction of the criteria.

ID	s	d	rank	VSD
13559	0	0	0	1
13398	1	1	1	1
13412	0	0	0	0,936
13428	1	0	1	0,742
13613	0	0	0	0,456
13790	0	1	-1	0,172
14268	0	0	0	0

Table 4.8: Satisfaction degree s, dissatisfaction degree d, value for BSD and value for VSD.

ID	$\omega = 0$	$\omega = 0.25$	$\omega = 0.5$	$\omega = 0.75$	$\omega = 1$
13559	1	0.75	0.5	0.25	0
13398	1	1	1	1	1
13412	0,936	0,702	0,468	0,234	0
13428	0,742	0,807	0,871	0,936	1
13613	0,456	0,342	0,228	0,114	0
13790	0,172	-0,121	-0,414	-0,707	-1
14268	0	0	0	0	0

Table 4.9: Comparative with different values for ω for result classification.

4.7. Conclusions

In this chapter, an overview and comparison of two commonly known approaches to bipolar querying of databases have been presented: the constraint-wish approach and the satisfied-dissatisfied approach. The specification of bipolar query conditions and different aspects of query handling, including the evaluation of elementary conditions, their aggregation, as well as ranking of the query results have been described. Also, the application of bipolar querying to temporal databases has been studied.

The constraint-wish approach has been specifically designed to cope with situations where user preferences express requirements —called constraints— which should be satisfied (at least to some extent) by the retrieved database tuples, and other, optional conditions —called wishes— which serve to distinguish among those tuples that satisfy the constraints to the same extent. Slightly different semantics is modelled by the 'and if possible' based approach to constraints and wishes, where the influence of the wishes on the results of a query depends on the existence of the tuples satisfying constraints and wishes at the same time.

The motivation for the satisfied-dissatisfied approach is to cope with user preferences that are composed of positive conditions —expressing what the user likes— and negative conditions —expressing what the user wants to avoid. The positive and negative conditions do not necessarily have to be complementary to each other.

Although both approaches result in pairs of satisfaction degrees (constraint satisfaction and wish satisfaction, or satisfaction degree and dissatisfaction degree), the semantics are quite different. In the constraint-wish approach, 'true' constraints, i.e., mandatory requirements, are treated as more important in a specific sense. In the satisfied-

dissatisfied approach, the positive and negative requirements are considered in general as being equally important and independent. Due to this assumed independence, it is also possible to model inconsistent or conflicting situations in the satisfied-dissatisfied approach, which is not possible in the constraint-wish approach, where either strong or weak consistency must apply. Moreover, in the satisfied-dissatisfied approach, the set of operators that can be used for ranking or aggregating is more elaborate than in the constraint-wish approach (e.g., weighted aggregation operators). On the other hand, a complete '*bipolar*' relational algebra has been developed for the constraint-wish approach, i.e., an extension of traditional relational algebra to handle bipolarity [189].

The introduction of temporal constraints in bipolar querying, leads to several issues related to the semantics of the constraints as well as the aggregation and ranking of the results. In this chapter we have presented two proposals to provide bipolar querying in a temporal database. Both are based on the satisfied-dissatisfied approach. In the first proposal, the temporal constraint is seen as a global constraint. The user can specify a temporal constraint which is specified by using one of the Allen relations [9] and a (possibilistic) time interval. The resulting tuples fulfil the bipolar constraints and the global temporal constraint. In this case the evaluation of the satisfaction and dissatisfaction degrees is a matter of aggregation between the bipolar satisfaction degree and the temporal satisfaction degree. The second proposal, allows to specify one or several temporal constraints in each elementary query conditions. In this case, the user can specify different temporal constraints for each attribute. This offers the user more expressive power, as shown in the examples of the criminal database.

In this chapter we presented a practical application of bipolar querying in a temporal database. The dataset is a real database SMLC. The most interesting characteristic of this database is the uncertainty in the temporal data. The information obtained by applying bipolar querying techniques, gives to the historian a classification of the query results in three groups: The results which satisfy the criteria, the results which do not satisfy the criteria at all and the indifferent results. This kind of information is not provided by the traditional querying techniques.

The novel contributions of this chapter are:

- The extension of the satisfied-dissatisfied approach for the bipolar querying of databases with temporal criteria.
- The aggregation and ranking between temporal criteria and non-temporal criteria.

Further research work will cover the specification, ranking and aggregation of bipolar temporal constraints in the satisfied-dissatisfied approach. In this case, the temporal constraint will be specified by using both positive and negative elementary temporal constraints.

Acknowledgements

Part of this research has been supported with the founding of the Hercules Foundation (Flanders) within the project "Sources from the Medieval Low Countries (SMLC)". We want to thanks Philippe Demonty for the dataset used in this chapter.

5

Visualization of Uncertain Time Intervals in the Triangular Model

The contents of this chapter have been partially published on:

- G. de Tré, A. Bronselaer, C. Billiet, Y. Qiang, N. Van De Weghe, P. de Maeyer, J. E. Pons, and O. Pons, "Visualising and handling uncertain time intervals in a two-dimensional triangular space," in *Proceedings of the 2nd World Conference on Soft Computing*, 12 2012.
- C. Billiet, J. E. Pons, O. Pons Capote, and G. Tré, "A comparison of approaches to Model Uncertainty in Time Intervals" in *Proceedings of the EUSFLAT conference*, Sep. 2013

Contents

Time modelling is an important issue in information management. Both traditional and soft computing methods have been thoroughly studied. Traditional approaches are specifically designed to efficiently deal with perfect temporal data. Soft computing techniques additionally support the efficient handling of imperfect temporal data. In this chapter, we study a novel soft computing technique to represent and handle time intervals that have an uncertain start and/or end point. Hereby, a two-dimensional representation, which is called the Triangular Model, is further generalized to efficiently cope with uncertain time intervals. The proposed approach provides a straightforward and compact visualisation for possibilistic interval data, which supports temporal reasoning and in which temporal distributions are easy to observe and to analyse.

5.1. Introduction

When using fuzzy sets to represent imperfect time intervals, one should take care about the underlying semantics [152]. Indeed, two main approaches exist and should be clearly distinguished.

First, a fuzzy set F can be used to model the time points that belong to a given imperfect time interval, e.g., all time points belonging to the 'industrial revolution'. For each time point t, the corresponding membership grade $\mu_F(t)$ then reflects the proximity of t to prototype elements of F . Membership grades are hence interpreted as degrees of similarity. The fuzzy set has a conjunctive interpretation: all its elements together represent the prototype. An example of such a fuzzy set is given in Fig 5.1(a). The fuzzy set representing 'industrial revolution' expresses for each time point, its proximity to the imperfect time interval known as 'industrial revolution'. For example, all dates between 1750 and 1850 fully belong to this period, while 1740 only belongs to an extent 0.4 to this period. Reasoning with 'fuzzy' time intervals under this conjunctive interpretation has, among others, been studied in [21], [23], [229].

Second, uncertainty about the start or end of a time interval can also be expressed by a fuzzy set, of which the membership function is interpreted as a possibility distribution. Under this interpretation, the membership grades of the membership function are interpreted as degrees of uncertainty. The fuzzy set has a disjunctive interpretation: it reflects all the possible candidates for the uncertain time point of which only one candidate is the actual value. If the same fuzzy set F as given in Fig 5.1(a) is used in this context to model the start point of a time interval, it denotes that the interval under consideration starts during the industrial revolution. The exact start date is not known with certainty, but modelled by a possibility distribution π_s which returns for each time point t, the possibility $\pi_s(t) = \mu_F(t)$. Only one time point with $\mu_F(t) \neq 0$ will turn out to be the actual start point of the interval. Using the same approach, a time interval with uncertain start point and end point can be modelled by two separate possibility distributions π_s and π_e as presented in Fig 5.1(b). The leftmost possibility distribution hereby reflects the uncertainty about the starting point of the interval, e.g., 'around 1750', whereas the rightmost possibility distribution reflects the uncertainty about the end point of the time interval, e.g., 'around 1850'. Hence the two membership functions in Fig 5.1(b) together represent an uncertain time interval which starts around 1750 and ends around 1850. This approach has been initially presented in [21] and fur-

ther developed within the context of temporal databases in [230], [181], as described in Chapter 3.

A problem with the presentation of uncertain time intervals as illustrated in Fig 5.1(b) is that two separate possibility distributions are used and presented along the same scale and in the same graphic. This might induce confusion and moreover does not allow it to directly derive the possibility that the effective time interval, e.g., 'starts in 1752 and ends in 1848'. To overcome these problems an alternative visualisation technique for uncertain time intervals is proposed in this chapter. The proposed approach is based on the time visualisation technique of Kulpa [231], [232], in which crisp time intervals are mapped to points in a two-dimensional space. Kulpa's model has been further developed, extended for analysis purposes and named the Triangular Model (TM) in [233]. In [234] an initial approach to handle imperfect time intervals in the TM represented by fuzzy sets with a conjunctive interpretation has been presented. In [235], [236] Qiang studied the modelling of imperfect time intervals in the TM using rough set theory. In this chapter, Qiang's rough set approach is further generalised to a fuzzy set based approach in which imperfect time intervals are represented by two fuzzy sets with a disjunctive interpretation.

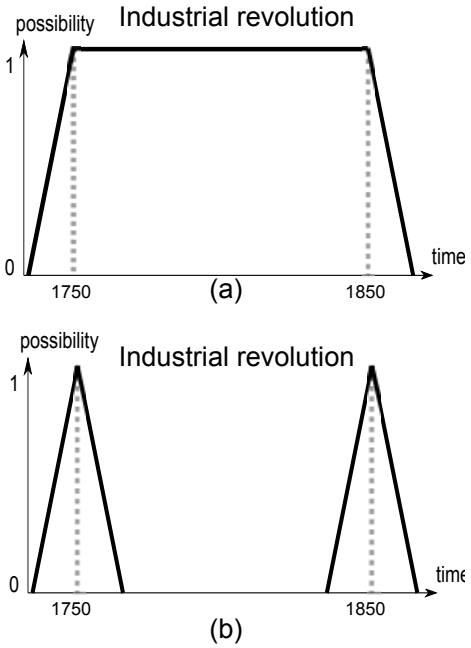


Figure 5.1: Different representations of fuzzy sets representing time intervals.

The remainder of the chapter is organised as follows. In the next Section 5.2 some preliminaries about the handling of uncertain time intervals and the triangular model (TM) are presented. In Section 5.3 it is studied how the TM can be used to represent uncertain time intervals. Next, the basic operational aspects of reasoning with uncertain time intervals in the TM are discussed in Section 5.4. Hereby, the thirteen Allen relations for crisp time intervals are generalised to handle uncertain time intervals and new methods for determining the relational information between an interval point and

an uncertain interval and between two uncertain intervals are proposed. Finally, the results of this work are discussed and some conclusions and issues for further research are presented in Section 5.6.

5.2. The triangular model

Two essential concepts are explained in the following two subsections. First an introduction to the uncertain time intervals (UTIs) and then an explanation of the triangular model.

5.2.1. Uncertain Time Intervals

As it has been shown before, in practice, it often occurs that time intervals cannot be exactly specified. Especially, in historical data it is often the case that either the starting date or the ending date (or both) of a time interval are partially or completely unknown. In such cases, the best thing to do is modelling the available knowledge about (the uncertainty of) the interval as accurate as possible, hereby avoiding information loss. Soft computing techniques, more specifically possibility theory [38], can be used for that purpose. In [230], [181] a possibilistic valid-time model able to cope with time intervals that have an uncertain start and/or end point has been presented in Chapter 3. The basics of this model are used and briefly described as follows. Each (Un)certain Time Interval (UTI) is defined by a pair

$$(\pi_s, \pi_e) \tag{5.1}$$

where π_s and π_e are two convex possibility distributions, respectively reflecting the knowledge about the start and end point of the UTI (as illustrated in Fig. 5.2(a)). In case of certainty, the possibility distribution is characterised by a fuzzy set with singleton support and core, containing the crisp date. In case of uncertainty, the possibility distribution is characterised by a normalized fuzzy set containing all possible candidate values for the date and their associated degree of possibility.

Together, both possibility distributions reflect the available knowledge about the start and end of the UTI they model. This implies that π_s and π_e together represent another possibility distribution π_I consisting of all possible time intervals that can be constructed from π_s and π_e . Some of the intervals in π_I are depicted in Fig. 5.2(b). For each time interval $[t_s, t_e]$, its associated possibility grade in π_I is computed by

$$\pi_I\left([t_s, t_e]\right) = \begin{cases} \min\left(\pi_s(t_s), \pi_e(t_e)\right), & \text{if } t_s \leq t_e \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Eq. (5.2) reflects that given the uncertainty about the start point, modelled by π_s and the uncertainty about the end point, modelled by π_e, the possibility that $[t_s, t_e]$ is the actual value of the UTI equals the possibility that t_s is the actual start point (i.e., $\pi_s(t_s)$) and t_e is the actual end point (i.e., $\pi_e(t_e)$) of I, where the conjunction of both conditions is modelled by the minimum t-norm. In the inconsistent case where $t_s > t_e$ the interval is considered to be completely impossible. In the following, we will denote an UTI as π_I, i.e., $\pi_I = (\pi_s, \pi_e)$.

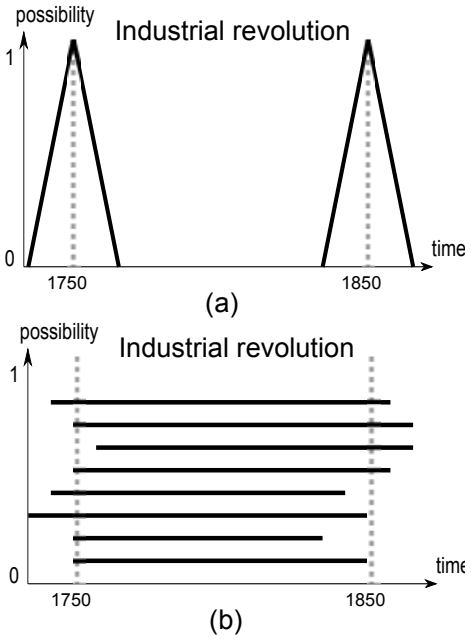


Figure 5.2: Possibilistic modelling of a time interval.

5.2.2. The triangular model

Mainly due to its linear characteristic, time is traditionally visualised using a linear representation where several time intervals are depicted as linear segments on top of a horizontal time axis. This approach has also been used in Fig. 5.2(b). One can read the start and end point of each interval on the horizontal scale. The vertical dimension is solely used to differentiate multiple overlapping intervals, if used at all. The linear representation works well as long as the number of represented time intervals remains low. As soon as a large number of overlapping intervals have to be represented the representation becomes overloaded and accurate visual data analysis becomes almost impossible.

For that reason, an alternative, Triangular time Model (TM) has been proposed in [233]. This model is an extension of Kulpa's triangular model for modelling crisp time intervals [231]. Basically, intervals are modelled as interval points in a two-dimensional space as follows. First, a horizontal time axis is considered. Second, for each time interval $[t_s, t_e]$ two straight lines L_s and L_e are constructed as depicted in Fig. 5.3. Line L_s is passing through t_s and makes a fixed angle α with the horizontal time axis, whereas L_e is passing through t_e making a fixed angle $-\alpha$ with the horizontal time axis. The interval $[t_s, t_e]$ is then represented by the intersection of L_s and L_e , which is called the interval point of $[t_s, t_e]$. For the ease of use, α is here chosen to be 45° . In TM all time intervals are represented by their corresponding interval point. So, because α is fixed for all interval points, the position of the interval point completely determines the start and end of the interval. The two-dimensional space of interval points is called the interval space and is denoted by $I\mathbb{R}$ [232].

From the observation that the start points and end points of two crisp time intervals

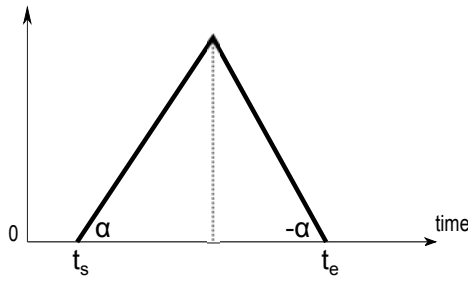


Figure 5.3: Construction of an interval point.

$I^1 = \left[t_s^1, t_e^1\right]$ and $I^2 = \left[t_s^2, t_e^2\right]$ can be smaller than ($<$), equal to ($=$) or larger than ($>$) each other, Allen proposed the thirteen possible relations between two crisp time intervals given in Table 5.1 [9].

In TM, these thirteen Allen relations each correspond to a specific zone in the interval space $I\mathbb{R}$ [231]. These thirteen zones are called Crisp Relational Zones (CRZs). The CRZs with respect to a reference interval point I are depicted in Fig. 5.4. Hereby, the following shorthand notations are used: equals (E), starts (S), started by (SB), finishes (F), finished by (FB), meets (M), met by (MB), overlaps (O), overlapped by (OB), during (D), contains (C), before (B) and after (A).

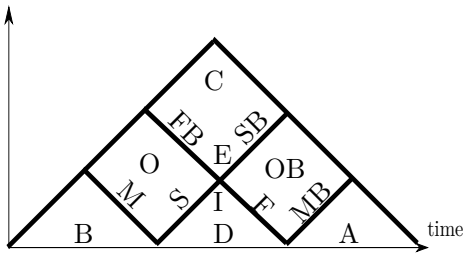


Figure 5.4: CRZs corresponding to the thirteen Allen relations with respect to the reference interval point I.

Each CRZ represents the set of interval points of the crisp time intervals that are in the corresponding Allen relation with respect to the reference interval I. For example, all interval points of intervals that come before I are located in the CRZ represented by the leftmost lower triangle in the interval space. Likewise, if an interval point is located in the top quadrangle above I, then its corresponding interval contains I. CRZs allow to visually analyse relations between (large) sets of interval points.

5.3. Representing Uncertain Time Intervals

In what follows, we propose a novel technique to model and visualise UTIs as introduced in Subsection 5.2.1 in the TM that has been presented in Subsection 5.2.2.

For each UTI π_I characterized by its two possibility distributions π_s and π_e , a corresponding Uncertain Interval Zone (UIZ) can be constructed as illustrated in Fig.5.5. The convex possibility distribution π_s determines the interval $[t_s^-, t_s^+]$ in which the start

Name	Implementation
I equals J	if $s_i = s_j \wedge e_i = e_j$
I starts J	if $s_i = s_j \wedge e_i < e_j$
I started by J	if $s_i = s_j \wedge e_i > e_j$
I finishes J	if $s_i > s_j \wedge e_i = e_j$
I finished by J	if $s_i < s_j \wedge e_i = e_j$
I meets J	if $e_i = s_j$
I met by J	if $s_i = e_j$
I overlaps J	if $s_i < s_j \wedge e_i < e_j \wedge e_i > s_j$
I overlapped by J	if $s_i > s_j \wedge e_j < e_i \wedge s_i < e_j$
I during J	if $s_i > s_j \wedge e_i < e_j$
I contains J	if $s_i < s_j \wedge e_i > e_j$
I before J	if $e_i < s_j$
I after J	if $s_i > e_j$

Table 5.1: Allen's relations represented in the framework. $I = [s_i, e_i]$, $J = [s_j, e_j]$

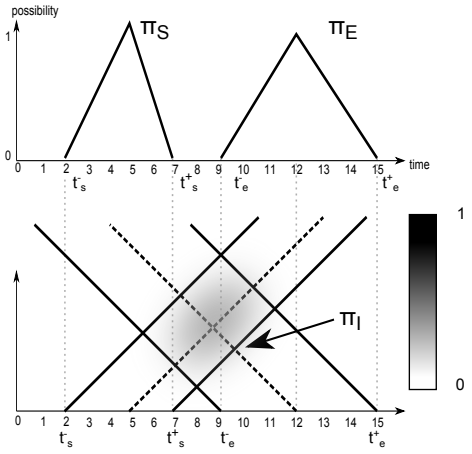


Figure 5.5: Construction of the UIZ corresponding to a given UTI π_I

point of the UTI is located, whereas the convex possibility distribution π_e determines the interval $[t_e^-, t_e^+]$ in which the end point of the UTI is located. Hence, all candidate time intervals for the UTI are contained within the interval $[t_s^-, t_e^+]$ (or during $[t_s^-, t_e^+]$) and contain the interval $[t_s^+, t_e^-]$. Using the CRZs that correspond to the Allen relations during and contains we obtain that the UTI must be located in the quadrangle zone that is determined by the two straight lines that respectively go through t_s^- and t_s^+ under an angle α and the two straight lines that respectively go through t_e^- and t_e^+ under an angle $-\alpha$. Not all interval points in this zone are to the same extent representing candidate intervals for the UTI. Indeed, by using Eq. (5.2) it is obtained that an interval $[t_s, t_e]$ is a candidate to the extent $\min(\pi_s(t_s), \pi_e(t_e))$ if $t_s \leq t_e$ and not a candidate at all else. To denote the extent to which an interval point is a candidate for the UTI, the point is colored using a grey scaled gradient color where black denotes 1 (completely possible candidate) and white denotes 0 (completely impossible candidate). The resulting

colored quadrangle zone is then called the UIZ of the UTI.

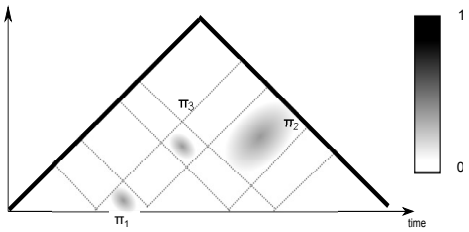


Figure 5.6: Using TM to represent UTIs.

Example 31. In Fig.5.6 some examples of UIZs are shown. UTI π_{I1}, π_{I2} and π_{I3} have both uncertain starting and ending points. From the position of the UTIs is possible to extract relational information. For example, it is clear that the uncertainty in the beginning of π_{I1} is related with π_{I3}. Also, π_{I3} is overlapped by (OB) π_{I2}. Finally, we can say that π_{I2} is after π_{I1}. In the following section we will study the temporal reasoning with uncertain time intervals.

5.4. Temporal reasoning with uncertain time intervals

In order to reason with UTIs, the thirteen CRZs for crisp interval points have been generalized. For a general UTI π_I , this generalisation resulted in fifteen so-called Uncertain Relational Zones (URZs), which are depicted in Fig. 5.7. These URZs are the basis for deriving relational information on interval points and UTIs with respect to the reference UTI. The positioning of interval points with respect to an UTI is discussed in the next Subsection 5.4.1, whereas the positioning of two UTIs is handled in Subsection 5.4.2.

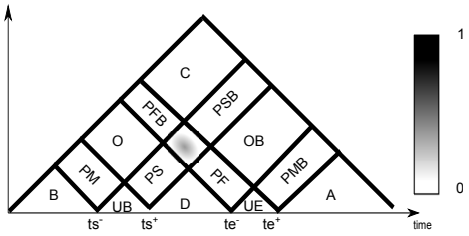


Figure 5.7: URZs: A generalisation of CRZs.

5.4.1. Relational Information for Interval Points

Like with CRZs, an URZ gives relational information on the positioning of an interval point with respect to the candidate interval points of an UTI. The naming and semantics of the fifteen URZs presented in Fig. 5.7 are described in Table 5.2. The first column contains the symbols used to represent the URZs, the second column gives the full name of the zone, whereas the third column sums up the different possible Allen relations that correspond to the URZ.

Symbol	Name	Relationships
B	Before	B
O	Overlaps	O
C	Contains	C
D	During	D
OB	Overlapped By	OB
A	After	A
PM	Possibly Meets	M, B, O
PS	Possibly Starts	O, S, D
PFB	Possibly Finished By	O, FB, C
PE	Possibly Equal	E, FB, SB, F, S, C, D,O, OB
PSB	Possibly Started By	C, OB, SB
PF	Possibly Finishes	OB, F, D
PMB	Possibly Met By	OB, MB A
UB	Uncertain Beginning	B, M, O, S, D
UE	Uncertain Ending	D, F, OB, MB, A

Table 5.2: The fifteen possible URZ for a given UTI.

The set of the fifteen relations corresponding to the fifteen URZs given in Table 5.2 is denoted R_U , i.e.,

R_U = { B, O, C, D, OB, A, PM, PS, PFB, PE, PSB, PF, PMB, UB, UE }.

Consider an interval point J in the interval space $I\mathbb{R}$. For the sake of generalisation it is assumed that J has an associated degree of possibility $\mu_{I\mathbb{R}}(J)$ which expresses to what extent it is possible that the interval point belongs to the interval space. (For interval points that represent a crisp interval it holds that $\mu_{I\mathbb{R}}(J) = 1$. For interval points belonging to an UIZ it more generally holds that $0 < \mu_{I\mathbb{R}}(J) \leq 1$, denoting to what extent it is possible that J is the actual interval of the UTI represented by the UIZ.)

Depending on the location of J with respect to the UTI π_I, we can have certainty or uncertainty about the possible Allen relations between J and π_I:

- If the interval point J is located in a zone with only one possible Allen relation. (From Fig. 5.7 and Table 5.2 it can be derived that this is the case if J is located in the URZ B, O, C, D, OB or A.) Then we know with certainty that J is in the corresponding Allen relation $R^{'}$ with UTI π_I . The relation is considered to be possible to an extent

$$\mu_{(JR\pi_I)}\left(R^{'}\right) = \mu_{I\mathbb{R}}\left(J\right) \tag{5.3}$$

  For all other (non-candidate) Allen relations the above degree of possibility is zero.

- If the interval point J is located in a zone with multiple candidate Allen relations. (This occurs when J is located in the URZ {PM , PS, PFB, PE, PSB, PF , PMB, UB or UE }.) Then there is uncertainty about the Allen relation that actually applies to J and the actual interval of the UTI π_I. The possible relations are those

mentioned in the Table 5.2. The uncertainties associated with these relations originate from the uncertainty in the UTI and can be computed as follows.

To start, draw two straight grid lines through J, reconstructing the triangle that determines the interval that is represented by J and draw two straight grid lines through the start and end point of this interval which are orthogonal to the sloping sides of the triangle. (These are the four dotted lines represented in Fig. 5.8.) A grid line can intersect with the UIZ of π_I. If this is the case, it subdivides the zone (or zones in case of multiple intersecting grid lines) of the UIZ into three sub zones: a sub zone at one side of the grid line, a sub zone at the other side of the grid line and a sub zone consisting of the segment of the grid line resulting from the intersection. As such, the grid lines subdivide $R^{'}$ the UIZ of π_I into as many different sub zones $\pi_I^{R'}$ as there are candidate Allen relations $R^{'}$ between J and π_I (Table 5.2). To identify the relation $R^{'}$ of a given $R^{'}$ sub zone $\pi_I^{R'}$, the CRZs of an arbitrary interval point of the sub zone are considered. $R^{'}$ is then the relation that corresponds to the CRZ in which J is located. $R^{'}$.

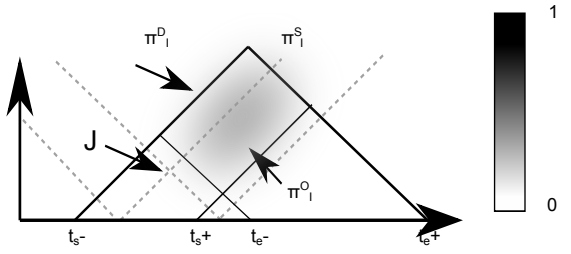


Figure 5.8: Determining the possible relation between an interval point J and UTI π_I.

Each sub zone $\pi_I^{R'}$ corresponds to a candidate Allen relation $R^{'}$ and contains those candidate interval points of π_I for which J is in relation $R^{'}$. The possibility that the candidate relation $R^{'}$ is the actual relation between J and π_I is then computed by

$$\mu_{IR\pi_I}\left(R^{'}\right) = \sup_{I \in \pi_I^{R'}} \min\left(\mu_{I\mathbb{R}}(J), \mu_I(I)\right) \tag{5.4}$$

Eq. (5.4) reflects that the possibility of $R^{'}$ equals the best of all possibilities that J belongs to the interval space and I represents the actual value of π_I . For all noncandidate Allen relations $R^{'}$, the sub zone $\pi_I^{R'}$ equals the empty set $\emptyset$, so Eq. (5.4) will return a possibility degree zero.

Hence, the possible relations $R^{'}$ between the interval point J and the UTI π_I can be modelled by the possibility distribution

$$\pi_{JR\pi_I} = \{\left(R^{'}, \mu_{JR\pi_I}\left(R^{'}\right)\right) | R^{'} \in R_U\} \tag{5.5}$$

where $\mu_{JR\pi_I}\left(R^{'}\right)$is computed using Eq. (5.3) or Eq. (5.4).

Example 32. To illustrate the determination of relational information for interval points, consider a situation as sketched in Fig. 5.8. The figure represents a quadrangle UIZ

corresponding to an UTI π_I and an interval point J, e.g., corresponding to a crisp interval. Because J is located in the URZ *possibly_starts(PS)*, J can either overlap (O) with the UTI π_I, be located during (D) π_I or start (S) at the same time as π_I.

To determine all the interval points of π_I with which J is in a given relation R, the four dotted grid lines are constructed. One grid line intersects with the UIZ of π_I O, S, D and subdivides it in three sub zones $\pi_I^O, \pi_I^S, \pi_I^D$ that respectively contain all interval points of π_I for which J is relation O, S and D. The interval points in each zone $\pi_I^{R'}, R' = O, S, D$ can then later be used to determine the possibility that J is in relation R with the UTI π_I (Eq. (5.4)).

5.4.2. Relational Information for UTIs

For the comparison of two UTIs π_J and π_I a generalisation of the technique presented in Subsection 5.4.1 can be used. Consider a reference UTI π_I. The same URZs as depicted in Fig. 5.7 can be constructed for the UIZ of π_I . The possible relations between a second UTI π_J and π_I can then be derived from the location of the UIZ of π_J with respect to URZs of π_I. This is illustrated in Fig. 5.9. Depending on how the UIZ of π_J is located with respect to the UIZ of π_I , we can have certainty or uncertainty about the applicable Allen relation between π_J and π_I:

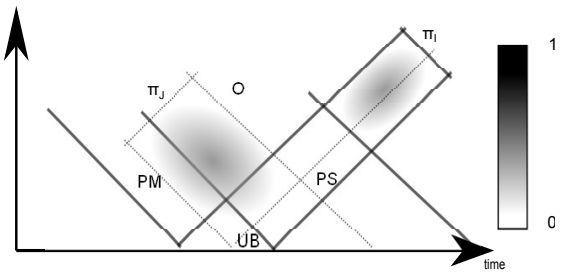


Figure 5.9: Determining the possible Allen relations between an UTI π_J and UTI π_I.

- If the UIZ of π_J is completely located in an URZ with only one possible Allen relation, i.e., if the intersection of the UIZ of π_J and URZ R' with $R' \in \{$ B, O, C, D, OB, A $\}$ equals the UIZ of π_J, then we know with certainty that the UTI π_J is in the Allen relation R' with the UTI π_I. This certainty follows from the fact that all candidate intervals for π_J are in Allen relation R' with all candidate intervals for π_I and R' is the only possible Allen relation for that UIZ. Hence the relation R' between π_J and π_I is then considered to be completely possible, i.e.,

$$\mu_{\pi_J R \pi_I}\left(R'\right) = 1 \tag{5.6}$$

  For all other (non-candidate) Allen relations the above degree of possibility is zero.

- In all the other cases, there is uncertainty about the Allen relation that applies between π_J and π_I. These other cases occur when the UIZ of π_J is completely

located in an URZ with multiple possible Allen relations (i.e., when the intersection of the UIZ of π_J and the URZ $R^{'}$ with $R^{'} \in \{$PM, PS, PFB, PE, PSB, PF, PMB, UB, UE $\}$ equals the UIZ of π_J) or when the UIZ of π_J intersects with multiple URZs, as illustrated in Fig. 5.9. In the cases of uncertainty the candidate Allen relations can be directly derived from the URZs that contain a part of the UIZ of π_J : all possible relations to π_I (as mentioned in Table 5.2) for the URZs under consideration define the set of candidate Allen relations.

Example 33. For example, the UIZ of π_J presented in Fig. 5.9 intersects with the URZs PM , O, PS and UB.

The sets of possible Allen relations corresponding to these URZs are:

- for PM : $\{$M, B, O$\}$;
- for O: $\{$O$\}$;
- for PS: $\{$O, S, D$\}$;
- for UB: $\{$B, M, O, S, D$\}$.

The set of candidate Allen relations between π_J and π_I is then obtained from the union of these sets, i.e., is the set $\{$B, M, O, S, D$\}$.

The uncertainty about which one of these relations $R^{'}$ actually applies, originates from the uncertainty about the actual interval of the UTI π_I and the uncertainty about the actual interval of the UTI π_J and can be computed from the possibility distributions $\pi_{JR\pi_I}$of all interval points J belonging to π_J (Eq. (5.5)).

Indeed, the possibility that the candidate relation $R^{'}$ is the actual relation between π_J and π_I is obtained by

$$\mu_{\pi_J R \pi_I}\left(R^{'}\right) = \sup_{J \in \pi_J} \pi_{JR\pi_I}\left(R^{'}\right) \tag{5.7}$$

and expresses that the best possibility of $R^{'}$ being the relation between an interval point J of π_J and the UTI π_I (seen over all interval points J of π_J) reflects the possibility that $R^{'}$ is the relation between π_J and π_I.

The possible relations $R^{'}$ between the UTI π_J and the UTI π_I can then be modelled by the possibility distribution

$$\pi_{\pi_J R \pi_I} = \left\{\left(R^{'}, \mu_{\pi_J R \pi_I}\left(R^{'}\right)\right) | R^{`'} \in R_U\right\} \tag{5.8}$$

where $\mu_{\pi_J R \pi_I}\left(R^{'}\right)$is computed using Eq. (5.6) or Eq. (5.7).

5.5. Link between TM and the IKC frameworks.

In this section, first the approaches of the IKC and TM frameworks towards interval representation are compared. Next, their approaches towards the evaluation of Allen relationships between two uncertain time intervals are compared.

5.5.1. Comparison of Approaches to Interval Representation

The ill-known constraint (IKC) framework uses ill-known time intervals (IKTI) to represent time intervals subject to uncertainty, whereas the TM framework uses UTI for this. In both approaches, to consider uncertainty about the exact crisp time intervals (CTI) which is intended, uncertainty about the exact starting and ending instants of the intended interval is considered and confidence in the context of this uncertainty is expressed using two possibility distributions. In an IKTI, these possibility distributions each define one ill-known value (IKV). One of these then defines the IKTI's starting instant and the other defines its ending instant. In an UTI, one of these possibility distributions is directly meant to define the UTI's starting instant and the other is directly meant to define the UTI's ending instant. It is obvious that the concepts of IKTI and UTI are exactly the same, except for the explicit usage of the concept of IKV in IKTI to define and describe uncertainty about starting and ending instants. This equality implies that both approaches have the same basic restriction: they cannot represent every kind of interval subject to uncertainty imaginable. For example, imagine an ordered set of instants coinciding with $\mathbb{N}_0$ and imagine an interval subject to uncertainty in this set, where the intended interval is either $[2, 4]$ (possibility 1) or $[3, 7]$ (possibility 1). This interval can not be modelled by a combination of possibility distributions defining the starting and ending instants without making $[2, 7]$ and $[3, 4]$ also possible as intended intervals to some extent. This issue was first suggested about IKTI in [230].

Now, a major difference between the two frameworks, concerning their approaches towards the representation of time intervals in general, is that the TM framework includes visualization in its approach.

A first consequence of this is that the TM framework allows the visualization of multiple CTI in the same image plane. In theory, it should also allow the visualization of multiple UTI in the same image plane. However, if the interval space area's visualizing these UTI overlap, it is not yet researched which greyscale (or other) color and intensity each interval point in this overlapping area should have, as the appearance of such interval point should both reflect the possibility of it being the interval intended by the first UTI and the possibility of it being the interval intended by the second UTI. The advantage of the ability to visualize multiple time intervals in the same image plane is that a human observer could easily assess certain characteristics of a distribution of time intervals from an image containing their visualization. Therefore, the TM framework provides an added value to the IKC framework since it supports the visualization of IKTI.

A second consequence of this is that the TM framework requires that an UTI can be visualized, using a visualization method which actually visualizes the possibility of a CTI of being the interval intended by this UTI for every CTI that has a non-zero possibility of this. Thus, a method is required (see (5.2)), to calculate the possibility of a given CTI of being the interval intended by an UTI based on the possibility distributions defining this UTI. This method is found by demanding that the given CTI's starting instant is the intended interval's starting instant *and* that the given CTI's ending instant is the intended interval's ending instant and by determining the possibility of the conjunction of both demands using the standard possibility theory conjunction operator 'minimum'. Although not necessary for the correct and consistent functioning

of the TM framework, it appears to be the intention of the TM framework that possibility about the exact starting or ending instant of the interval intended by the UTI could be derived from the possibility distribution defining the possibility that a given CTI is the interval intended by the UTI. For this derivation to be consistent, the possibility distributions defining the UTI must be convex [177]. Given an ordered set T and an UTI $J = (\pi_{J_s}, \pi_{J_e})$ in T with possibility $\pi_J(I)$ that a given CTI $I = [s_i, e_i]$ is the exact time interval intended by J, the derivations can be calculated as follows:

$$\pi_{J_s}(s_i) = \max_{K=[s_i,k],k\in T,k>s_i}(\pi_J(K))$$
$$\pi_{J_e}(e_i) = \max_{K=[k,e_i],k\in T,k<e_i}(\pi_J(K))$$

With respect to this convexity demand, the IKC framework is similar: the possibility distributions defining the starting and ending IKV of an IKTI are also demanded to be convex by the IKC framework, although this appears not to be necessary for the framework to function correctly and consistently.

5.5.2. Comparison of Approaches to Allen Relationship Evaluation

As mentioned before, a major difference between the two frameworks, concerning their approaches towards the representation of time intervals in general, is that the TM framework includes visualization in its approach. As a result, it also includes visualization in its approach towards the evaluation of Allen relationships between a CTI and an UTI.

A first consequence of this is that, given a CTI, an UTI, its URZ and their visualizations in the same image, the TM framework allows a visual, human assessment of the degree of possibility of this CTI being in an Allen relationship with this UTI, for every Allen relationship with a non-zero such degree, based on this image. Moreover, those with possibility degree equal to zero can be easily found before any calculation is done: they are the relationships not contained in the set corresponding to the URZ containing the CTI's interval point. On the other hand, to examine the respective possibilities of a given CTI of being in several different Allen relationships with a given IKTI using the IKC framework, a new collection of specific IKC and a specific aggregation of these should be constructed for every Allen relationship under consideration, allowing to calculate its exact possibility and necessity. In contrast to the TM framework, using the IKC framework a human assessment before any calculation is not possible and it is not known before any calculation which Allen relationships will result in a possibility degree equal to zero.

A second consequence is that, given an UTI and its URZ, multiple CTI can be visualized in the same image as the UTI and its URZ. Thus, the same image could provide a visual, human assessment of the possibilities with which multiple CTI are in Allen relationships with the UTI, before any calculation is done. Again, the IKC framework would need a different collection of specific IKC and a specific aggregation of these for every Allen relationship under consideration, but calculating the possibilities for several CTI to be in a given Allen relationship with a given IKTI would not provide much extra work.

A third consequence is that, in the TM framework, given an UTI, its URZ and a visualization of these in the same image, a given Allen relationship could correspond with several URZ. Thus, given a distribution of CTI, it is not easy to visually assess which CTI have a non-zero possibility of being in the given Allen relationship with the given UTI. On the other hand, assessing this using the IKC framework is impossible without any calculation, but the necessary calculations are pretty straightforward.

Although this has not been rigorously researched yet, it is the conviction of the authors that a great strength of the IKC framework lies in its modular approach towards the evaluation of temporal relationships, resulting in a flexibility and an easy handling of complex temporal relationships, while the visualization of these using the TM framework could become complex and heavy. This would give the IKC framework a major advantage over the TM framework, when used in reasoning systems like e.g. decision support systems. Moreover, in some cases the visualization step used in the TM framework could be a redundant step.

Example 34. Consider an ordered set of instants T coinciding with $\mathbb{R}$, a CTI $I = [1, 3]$ and a time interval J of which the starting instant is defined by a triangular possibility distribution on T with core $\{5\}$ and support $[2, 7]$ and of which the ending instant is defined by a triangular possibility distribution on T with core $\{12\}$ and support $[9, 15]$. The visualization of this example situation using the TM framework is shown in Figure 5.10. The interval point for I lies in the URZ 'PM'. Thus, possibility is higher than zero for I to be in a 'meets', 'before', or 'overlaps' relationship with J. As the darkest point in the 'overlaps' area part is very light, the darkest point in the 'meets' line segment is of almost exactly the same lightness and the darkest point in the 'before' area part is perfectly black, it can be estimated that I overlaps J with a low possibility, I meets J with the same low possibility and I is before J with possibility 1. Calculation using the IKC framework now learns that:

$$\begin{aligned} \mathrm{Pos}(IoverlapsJ) &= \min(1, 1/3, 1) = 1/3 \\ \mathrm{Pos}(ImeetsJ) &= \min(1/3, 1) = 1/3 \\ \mathrm{Pos}(IbeforeJ) &= \min(1) = 1 \end{aligned}$$

5.6. Conclusions

Uncertain time intervals are time intervals for which there is uncertainty about the start and / or end point (or both). A possible approach to model uncertain time intervals is to use two possibility distributions: one reflecting the uncertainty about the actual start point and the other reflecting the uncertainty about the actual end point of the interval. Both possibility distributions together then form a possibility of candidate intervals (of which one is the unknown actual value of the uncertain time interval). Due to their large quantity, representing the candidate intervals of an uncertain time interval using a traditional linear time model is not straightforward, nor can be done efficiently. Moreover in case of multiple coexistent uncertain time intervals, such a linear representation is almost not accessible for further visual analysis. For these reasons, a novel

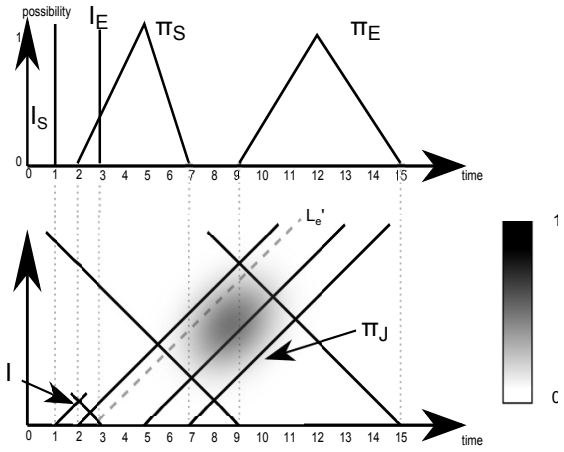


Figure 5.10: The visualization of the example using the TM framework.

modelling technique for uncertain time intervals has been proposed in this chapter. This technique is a generalisation of the triangular time modelling approach that originally has been presented by Kulpa. In the generalisation, each candidate time interval of an uncertain time interval is represented as an interval point in a twodimensional triangular space. The interval point is colored using a grey scaled gradient color representing the possibility ($\in [0, 1]$) that the candidate interval is indeed the actual unknown interval. Hence, an uncertain time interval is represented by a colored zone in the triangular space, which is called the uncertain interval zone of the uncertain interval. Multiple uncertain time intervals each have their own uncertain interval zone, which might be overlapping with other uncertain interval zones. Next to the representation of uncertain time intervals, the determination of their positioning related to each other has been studied. For that purpose fifteen uncertain relational zones related to an uncertain interval zone have been proposed. These zones are generalisations of the thirteen relational zones corresponding to the thirteen Allen relations for crisp time intervals introduced by Kulpa. Each uncertain relational zone corresponds to a set of candidate Allen relations. An interval point that is located in such a zone can then be in one of these candidate relations with the uncertain time interval. Using uncertain relational zones two novel techniques respectively determining the candidate Allen relations between an interval point and an uncertain time interval and two uncertain time intervals have been proposed. Where the uncertain interval zones permit to visually determine the possible Allen relations between two uncertain time intervals, the proposed techniques allow to compute their corresponding possibility degrees.

On the other hand, we have compared the two different frameworks designed to represent time intervals subject to uncertainty and evaluate temporal relationships between such intervals and crisp intervals: the triangular model (TM) framework and the ill-known constraint (IKC) framework. It is concluded that:

- With respect to representation, both frameworks differ only slightly, with the TM framework allowing easier human assessments, due to its approach including visualization.

- With respect to temporal relationship evaluation, the TM framework allows easy human assessments in several situations, but the IKC framework seems more fitted for complex reasoning, due to its modular build.

The novel contributions of this chapter are:

- The visualization of uncertain time intervals in the triangular model.
- The calculation of the relational information between two uncertain time intervals.

In future research we plan to investigate the applicability of the presented techniques in flexible querying of temporal databases containing uncertain interval data. Research topics that are of interest in this respect are: (1) the handling of elementary query conditions based on the fifteen generalised Allen relations corresponding to the uncertain relational zones, (2) the use of additional interval analysis operations like in between (two given uncertain time intervals), starts within (a given uncertain time interval) and ends within (a given uncertain time interval) and operations that put constraints on the duration of uncertain time intervals and (3) the handling of composite query conditions. Other planned activities include the implementation of a prototype software and the study of performance issues, including indexing.

6

An Open Source Framework for Fuzzy Temporal Databases

The contents of this chapter have been partially published on:

- J. E. Pons, I. Blanco Medina, and O. Pons Capote, "Generalised fuzzy types and querying: implementation within the hibernate framework," in *Proceedings of the 9th international conference on Flexible Query Answering Systems*, FQAS'11, (Berlin, Heidelberg), pp. 162–173, Springer-Verlag, 2011.
- J. E. Pons, O. Pons Capote, and I. Blanco Medina, "A fuzzy valid-time model for relational databases within the hibernate framework," in *Proceedings of the 9th international conference on Flexible Query Answering Systems*, FQAS'11, (Berlin, Heidelberg), pp. 424–435, Springer-Verlag, 2011.
- J. E. Pons, O. Pons, and I. Blanco Medina, "An open source framework for fuzzy representation and querying in fuzzy databases," in *Proceedings of the IADIS International Conference Informations Systems* (P. P. Miguel Baptista Nunes, Pedro Isaías, ed.), March 2011.
- J. E. Pons, O. Pons, and I. B. Medina, *New trends on intelligent systems and soft computing*, ch. Fuzzy temporal information treatment in relational DBMS. Theoretical Formulation, Implementation and Applications, pp. 95–112.

Contents

In this chapter we will implement a relational database system to deal with both imperfections in representation and flexible querying of temporal databases, using our possibilistic approach, presented in chapter 3. Despite of other relational temporal database implementations made in commercial systems like Oracle Workspace Manager, we want to obtain an open source and multiplatform implementation. To achieve this, we work with the Hibernate Framework [237] an open source object-relational tool. It is written in Java, hence it is multiplatform. And it allows to switch the underlying database for an application by just changing a configuration file. We will modify the source code of this framework to deal with fuzzy representation and querying first and second, to deal with the possibilistic valid-time model presented before.

The present chapter is organized as follows: Section 6.1 introduces the Hibernate Framework. The architecture of the framework is described in detail. The workflow for making objects persistent is also explained. Then the different methods provided in the framework for querying are introduced. Next to that, Section 6.2 presents the criteria and the strategies that were followed for the implementation within the Hibernate Framework. Section 6.3 shows a comparison among the proposals that can be found in the literature. The conclusions and the benefits of each approach are presented in Section 6.4.

6.1. Hibernate Framework

The Hibernate Framework [237] is a collection of open source projects that enable developers to make object-relational mappings. The framework needs an object-oriented language (Java) and a relational database management system, DBMS. A query language called Hibernate Query Language (*HQL*) is also provided, which is an object-oriented extension to SQL.

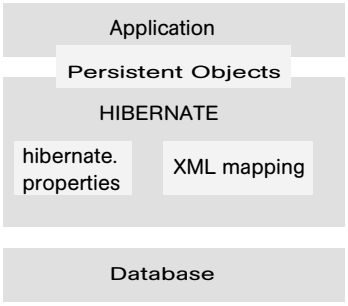


Figure 6.1: High level Hibernate architecture.

6.1.1. Architecture

An application developed to work with the Hibernate Framework has three layers as shown in Figure 6.1:

1. **Application layer**: This is the top layer. The application makes CRUD (CReate,Update and Delete) operations by means of persistent objects: an object in

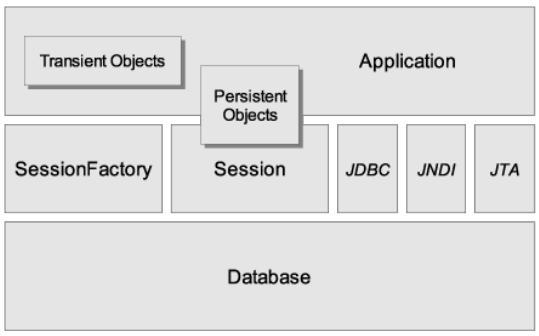


Figure 6.2: Detailed Hibernate with lite configuration.

the application representing an entity in the database. There exist three states for an object:

a) Persistent: The object represents a current database state: the object and the database entity are linked by a database session.

b) Transient: The object is no longer attached to a database session. The object and the database entity are not linked.

c) Detached: A detached instance is an object that has been persistent but its database session has been closed.

2. **Hibernate layer**: This layer acts as an abstraction layer between the DBMS and the application. The framework introduces the concept *dialect*: A dialect is an abstraction for the specific DBMS. Thus, the application will work with any DBMS by changing the dialect.

3. **The database**: This layer is the DBMS. The Hibernate Framework supports most of the commercial DBMS in the market. Each DBMS has it own dialect. Therefore, MySQL, PostgreSQL, Oracle and many other DBMS have a specific dialect.

A deeper look into the architecture shows two main configurations. The first is a lite configuration where the application manages the JDBC[1] connections. The second is a comprehensive configuration and allows Hibernate to take care of the JDBC and JTA[2] API, so the application is abstracted from that kind of details. Both configurations are illustrated in Figures 6.2 and 6.3 respectively.

Here are some brief discussions about some of the API objects depicted in the preceding illustrations:

[1]Java Database Connectivity [238]:The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.

[2]Java Transaction API [239]: Java Transaction API (JTA) specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.

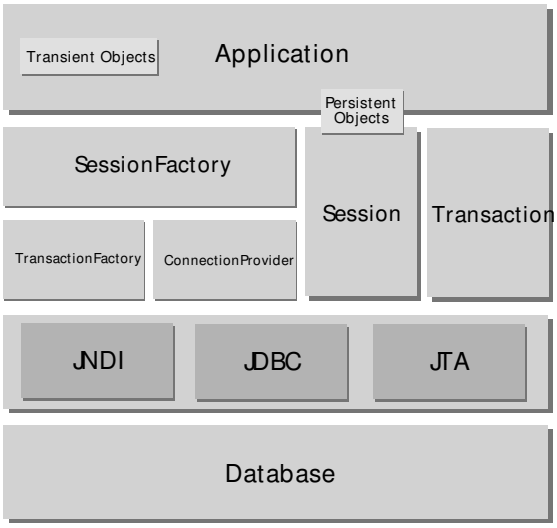


Figure 6.3: Detailed Hibernate with the comprehensive configuration.

- **SessionFactory (**`org.hibernate.SessionFactory`**)** A thread-safe, immutable cache of compiled mappings for a single database. A factory for `org.hibernate.Session` instances. A client of `org.hibernate.connection.ConnectionProvider`. Optionally maintains a second level cache of data that is reusable between transactions at a process or cluster level.

- **Session (**`org.hibernate.Session`**)** A single-threaded, short-lived object representing a conversation between the application and the persistent store. Wraps a JDBC `java.sql.Connection`. A factory for `org.hibernate.Transaction`. Maintains a first level cache of the application's persistent objects and collections; this cache is used when navigating the object graph or looking up objects by identifier.

- **Persistent objects and collections** Short-lived, single-threaded objects containing a persistent state and business function. These can be ordinary JavaBeans[3]/ POJOs[4]. They are associated with exactly one `org.hibernate.Session`. Once the `org.hibernate.Session` is closed, they will be detached and free to use in any application layer (for example, directly as data transfer objects to and from presentation). The difference among transient, persistent and detached object states is discussed in [242].

[3] A JavaBean [240] is a POJO that is serializable, has a no-argument constructor, and allows access to properties using getter and setter methods that follow a simple naming convention. Because of this convention, simple declarative references can be made to the properties of arbitrary JavaBeans. Code using such a declarative reference does not have to know anything about the type of the bean, and the bean can be used with many frameworks without these frameworks having to know the exact type of the bean.

[4] Plain Old Java Objects [241]: The term "POJO" is mainly used to denote a Java object which does not follow any of the major Java object models, conventions, or frameworks. The term continues the pattern of older terms for technologies that do not use fancy new features. Ideally speaking, a POJO is a Java object not bound by any restriction other than those forced by the Java Language Specification. (For example, a POJO should not have to extend or implement prespecified classes, neither contain prespecified annotations.)

ID	Legal Entity	Publication Date
54	N.A.T.O	22 / 10 / 2012
55	E.U.	14 / 11 / 2012
56	C.E.I.	25 / 9 / 2012

Table 6.1: Example of diplomatic document database.

- **Transient and detached objects and collections** Instances of persistent classes that are not currently associated with a `org.hibernateSession`. They may have been instantiated by the application and not yet persisted, or they may have been instantiated by a closed `org.hibernate`. See [242].
- **Transaction (`org.hibernate.Transaction`)**
  (Optional) A single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC, JTA or CORBA[5] transaction. An `org.hibernate.Session` might span several `org.hibernate.Transactions` in some cases. However, transaction demarcation, either using the underlying API or `org.hibernate.Transaction`, is never optional.
- **ConnectionProvider (**
  `org.hibernate.connection.ConnectionProvider`**)** (Optional) A factory for, and pool of, JDBC connections. It abstracts the application from underlying Datasource or DriverManager (`javax.sql.DataSource`, `java.sql.DriverManager`). It is not exposed to application, but it can be extended and/or implemented by the developer.
- **TransactionFactory**
  (`org.hibernate.TransactionFactory`**)** (Optional) A factory for `org.hibernate.Transaction` instances. It is not exposed to the application, but it can be extended and/or implemented by the developer.
- **Extension Interfaces** Hibernate offers a range of optional extension interfaces you can implement to customize the behavior of your persistence layer. See the API documentation [237] for details.

The mentioned elements are illustrated within an example.

Example 35. Consider a database with diplomatics documents. For each document, the following information is stored: the identifier of the document (field ID) the name of the entity that produces the document and the day that the document was published. An example instance of this database is shown in Table 6.1. The user wants to make the following query:

"Select the document with ID='56' and change the name of the legal entity to 'N.A.T.O' "

[5]Common Object Request Broker Architecture [243, 244] (CORBA): is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together (i.e., it supports multiple platforms).

We will illustrate the workflow to accomplish this task within the Hibernate Framework. First of all, it is necessary to define an entity class in Java to represent a diplomatic document object.

```
/**
 * Entity class for a diplomatic document.
 *
 */
@Entity // Java annotation to indicate Hibernate
    // that this class is an entity class

public class Document implements Serializable {
    // Annotations to set the generated identifier.
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Basic(optional = false)
    @Column(name = "ID")
    private Integer id;
    @Basic(optional=false)
    @Column(name = "LegalEntity")
    private String legalEntity;
    @DateTime
    @Column(name = "PublicationDate")
    private Date publicationDate;
/** Rest of the code:
getters and setters, constructor, etc  */
...
}
```

Then, the code to initialize the Hibernate framework and to make the query is the following:

```
Document doc = new Document(); // A transient instance
/* Obtaining the session from session factory */
Session session = HibernateUtil.getSession();
/* Obtaining the transaction */
Transaction t = session.beginTransaction();
long id = 56;
/* load the object */
session.load(doc, id);
/* update the field */
doc.setLegalEntity("N.A.T.O.");
/* Store the object */
session.save(doc);
/* commit the transaction */
t.commit();
/* close the session */
session.close();
/* at this point, the object doc is a detached object */
```

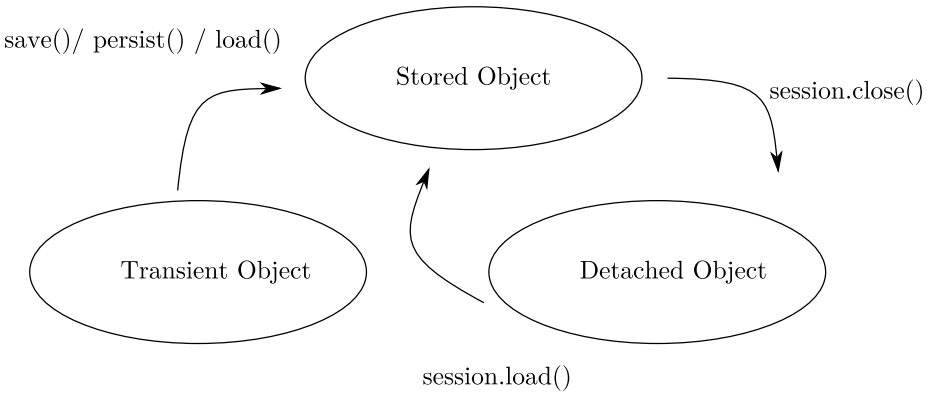


Figure 6.4: Transition among the different Hibernate objects states

6.1.1.1. Hibernate objects states

In this subsection we are going to define in depth the main object states, which are illustrated in figure 6.4:

- **Transient** - an object is transient if it has just been instantiated using the new operator, and it is not associated with a Hibernate Session. It has no persistent representation in the database and no identifier value has been assigned. Transient instances will be destroyed by the garbage collector if the application does not hold a reference anymore. Use the Hibernate Session to make an object persistent (and let Hibernate take care of the SQL statements that need to be executed for this transition).
- **Persistent** - a persistent instance has a representation in the database and an identifier value. It might just have been saved or loaded, however, it is by definition in the scope of a Session. Hibernate will detect any changes made to an object in persistent state and synchronize the state with the database when the unit of work completes.
- **Detached** - a detached instance is an object that has been persistent, but its Session has been closed. The reference to the object is still valid and the detached instance might even be modified in this state. A detached instance can be reattached to a new Session at a later point in time, making it (and all the modifications) persistent again. This feature enables a programming model for long running units of work that require user think-time. We call them application transactions, i.e., a unit of work from the point of view of the user.

6.1.1.2. Making objects persistent

Newly instantiated instances of a persistent class are considered transient by Hibernate. We can make a transient instance persistent by associating it with a session:

```
Document doc = new Document();
doc.setLegalEntity("N.A.T.O.");
```

```
doc.setPublicationDate("25/11/2012");
Long generatedId = (Long) session.save(doc);
```

If *Doc* has a generated identifier, the identifier is generated and assigned to the document when `save()` is called. If *Doc* has an assigned identifier, or a composite key, the identifier should be assigned to the document instance before calling save(). You can also use `persist()` instead of `save()`, with the semantics defined in the EJB3[6] specification.

`persist()` makes a transient instance persistent. However, it does not guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time. `persist()` also guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. This is useful in long-running conversations with an extended Session/persistence context. `save()` guarantees to return an identifier. If an INSERT has to be executed to get the identifier, this INSERT happens immediately, no matter if you are inside or outside a transaction. This is problematic in a long-running conversation with an extended Session/persistence context. Alternatively, you can assign the identifier using an overloaded version of `save()`.

Example 36. Consider the database of diplomatic documents in the previous example. In this case, the documents have a collection of associated publications. The user wants to save the object with its corresponding associated objects. These objects can be made persistent in any order you like unless you have a NOT NULL constraint upon a foreign key column. There is never a risk of violating foreign key constraints. However, you might violate a NOT NULL constraint if you `save()` the objects in the wrong order.

```
Document pk = new Document();
pk.setLegalEntity("E.U.");
pk.setPublicationDate("25/11/2012");
pk.setAssociatedDocuments( new HashSet() );
pk.addDocument(doc);
session.save( pk, new Long(1234) );
```

Usually the user does not bother with this detail, as you will normally use Hibernate's transitive persistence feature to save the associated objects automatically. Then, even NOT NULL constraint violations do not occur - Hibernate will take care of everything. Transitive persistence is discussed later in this chapter.

6.1.1.3. Loading an object

The `load()` methods of Session provide a way of retrieving a persistent instance if you know its identifier. `load()` takes a class object and loads the state into a newly instantiated instance of that class in a persistent state.

```
Document doc =
(Document) session.load(Document.class, generatedId);
```

[6] Enterprise JavaBeans (EJB) [245, 246] is a managed, server-side component architecture for modular construction of enterprise applications. The EJB specification is one of several Java APIs in the Java EE specification. EJB is a server-side model that encapsulates the business logic of an application.

```
// you need to wrap primitive identifiers
long id = 1234;
Document pk =
(Document) sess.load( Document.class, new Long(id) );
```

Alternatively, you can load the state into a given instance:

```
Document doc = new Document();
// load pk's state into document
session.load( doc, new Long(pkId) );
Set associatedDocuments = doc.getAssociatedDocuments();
```

Be aware that `load()` will throw an unrecoverable exception if there is no matching database tuple. If the class is mapped with a proxy, `load()` just returns an uninitialized proxy and does not actually hit the database until you invoke a method of the proxy. This is useful if you wish to create an association to an object without actually loading it from the database. It also allows multiple instances to be loaded as a batch if batch-size is defined for the class mapping.

If you are not certain that a matching tuple exists, you should use the `get()` method which hits the database immediately and returns null if there is no matching tuple.

```
Document doc =
  (Document) session.get(Document.class, id);
if (doc==null) {
    doc = new Document();
    session.save(doc, id);
}
return doc;
```

You can even load an object using an SQL SELECT ... FOR UPDATE, using a LockMode. We refer to the API documentation for more information.

```
Document doc = (Document) session.
get(Document.class, id, LockMode.UPGRADE);
```

Any associated instances or contained collections will not be selected FOR UPDATE, unless you decide to specify lock or all as a cascade style for the association.

It is possible to re-load an object and all its collections at any time, using the refresh() method. This is useful when database triggers are used to initialize some of the properties of the object.

```
sess.save(doc);
sess.flush(); //force the SQL INSERT
//re-read the state (after the trigger executes)
sess.refresh(doc);
```

How much does Hibernate load from the database and how many SQL SELECTs will it use? This depends on the fetching strategy. This is explained in [247] (Section 20.1, "Fetching strategies").

6.1.2. Querying in the Hibernate framework

Hibernate provides four methods for querying. Hibernate supports an easy-to-use but powerful object oriented query language Hibernate Query Language (HQL) [248]. For programmatic query creation, Hibernate supports a sophisticated Criteria and Example query feature (QBC[7] and QBE[8]). It is also possible to express a query in native SQL language of the underlying database, with optional support from Hibernate for resultset conversion into objects.

Example 37. Consider the diplomatic document database in the previous examples. The user wants to obtain the document published on '25/22/2012'. This can be expressed using any of the methods explained previously. Ordered from the more object-oriented approach to the more relational approach:

1. **Query by criteria**: Selects objects that fulfil a set of criterion. E.g.:

```
createCriteria(Document.class).
Add(Restrictions.
eq("PublicationDate","25/11/2012"));
```

2. **Query by example**: Selects objects similar to a given object. E.g.:

```
Document doc.PublicationDate =
new Date("25/11/2012");
```

3. **HQL**: An object-oriented language based on SQL. E.g.:

```
Query q = session.createQuery("SELECT doc
FROM Document doc
WHERE doc.publicationDate  = :publication ");
q.setDate("25/11/2012","publication");
```

4. **SQL**: A SQL sentence. E.g.:

```
"SELECT *
FROM Document as doc
WHERE doc.publicationDate"  = "25/11/2012" ;
```

It is important to notice that each type of query is translated into an SQL query. That sentence is customized for the underlying database by translating it into the database dialect.

[7]Query by criteria [249]: The Criteria API is used to define queries for entities and their persistent state by creating query-defining objects. Criteria queries are written using Java programming language APIs, are typesafe, and are portable. Such queries work regardless of the underlying data store.

[8]Query By Example: [250] is a database query language for relational databases. The motivation behind QBE is that a parser can convert the user's actions into statements expressed in a database manipulation language, such as SQL

6.1.2.1. Executing queries

HQL and native SQL queries are represented with an instance of `org.hibernate.Query`. This interface offers methods for parameter binding, for resultset handling, and for the execution of the actual query. The user always obtains a Query using the current Session:

```
List Documents = session.createQuery(
"from Documents as doc" +
"where doc.publicationDate < ?")
.setDate(0, date)
.list();
```

A query is usually executed by invoking list(). The result of the query will be loaded completely into a collection in memory. Entity instances retrieved by a query are in a persistent state. The uniqueResult() method offers a shortcut if you know your query will only return a single object. Queries that make use of eager fetching of collections usually return duplicates of the root objects, but with their collections initialized. You can filter these duplicates through a Set.

6.1.2.2. Iterating results

Occasionally, better performance might be achieved by executing the query using the `iterate()` method. This will usually be the case if we expect that the actual entity instances returned by the query will already be in the session or second-level cache. If they are not already cached, `iterate()` will be slower than `list()` and might require many database hits for a simple query, usually one for the initial select which only returns identifiers, and n additional selects to initialize the actual instances.

```
// fetch ids
Iterator iter =
sess.createQuery(
"from Documents d order by d.publicationDate")
.iterate();
while ( iter.hasNext() ) {
    Qux qux = (Qux) iter.next();  // fetch the object
    // something we couldnt express in the query
    if ( d.calculateComplicatedAlgorithm() ) {
        // delete the current instance
        iter.remove();
        // dont need to process the rest
        break;
    }
}
```

Queries that return tuples Hibernate queries sometimes return tuples of objects. Each tuple is returned as an array:

```
Iterator documentsAndAssociated =
session.createQuery(
```

```
"select doc, associated from Documents doc
join doc.associatedDocuments associated")
          .list()
          .iterator();

while ( documentsAndAssociated.hasNext() ) {
    Object[] tuple =
    (Object[]) documentsAndAssociated.next();
    Document doc = (Document) tuple[0];
    Set associated = (Set) tuple[1];
    ....
}
```

Scalar results Queries can specify a property of a class in the select clause. They can even call SQL aggregate functions. Properties or aggregates are considered ”scalar” results and not entities in persistent state.

```
Iterator results = session.createQuery(
"select min(doc.publicationDate), count(doc)
from Document doc group by doc.entity")
        .list()
        .iterator();

while ( results.hasNext() ) {
    Object[] tuple = (Object[]) results.next();
    Date oldest = (Date) tuple[1];
    Integer count = (Integer) tuple[2];
    .....
}
```

Bind parameters Methods on Query are provided for binding values to named parameters or JDBC-style ’?’ parameters. Contrary to JDBC, Hibernate numbers parameters from zero. Named parameters are identifiers of the form *:name* in the query string. The advantages of named parameters are as follows:

- named parameters are insensitive to the order they occur in the query string.
- they can occur multiple times in the same query.
- they are self-documenting.

```
//named parameter (preferred)
Query q = session.createQuery(
"from Document doc
where doc.legalEntity = :name");
q.setString("name", "E.U.");
Iterator docs = q.iterate();
//positional parameter
Query q = session.createQuery(
```

```
"from Document doc where doc.name = ?");
q.setString(0, "N.A.T.O.");
Iterator docs = q.iterate();
//named parameter list
List names = new ArrayList();
names.add("E.U.");
names.add("N.A.T.O.");
Query q = session.createQuery
("from Document doc
 where doc.legalEntity in (:namesList)");
q.setParameterList("namesList", names);
List docs = q.list();
```

6.2. The stratified model

This section explains the stratified architecture for a fuzzy database within the Hibernate Framework. First of all, we will describe the design and main criteria for the portability of a fuzzy representation and query system. Next, the main architecture of the system is explained in detail.

6.2.1. A Stratified Architecture

The use of a stratified architecture has been introduced previously in the literature. In [251], the authors propose a stratified model to implement a temporal database management system (DBMS) on the top of a relational database. The Hibernate framework fits really well with a stratified architecture.

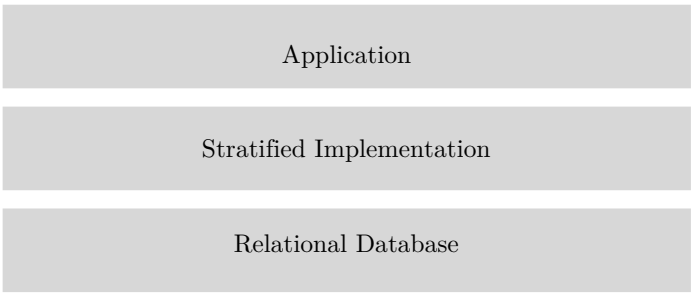


Figure 6.5: Stratified architecture.

The main idea for a stratified architecture is illustrated in figure 6.5. In the stratified approach, the applications are not connected directly to the DBMS. Hence, the communication between the application and the DBMS is done through the stratum. The main advantages when using a stratum are:

- It is possible to provide applications with a different data model than is actually implemented.
- The new data model does not have to be supplied by the DBMS vendor, therefore, it is possible to use any relational DBMS.

When this approach is used, the idea is to convert a relational DBMS which supports a SQL standard into a possibilistic temporal database. When an application sends a possibilistic temporal query to the database. The queries are translated in the stratum (the Hibernate layer) and converted into SQL queries.

6.2.2. Design Criteria

In order to achieve the goals for the implementation, it is necessary to specify a set of design requirements:

- **No changes** The proposed implementation shall not require any changes to the underlying database. Neither for the representation of fuzzy types nor for the fuzzy querying by means of fuzzy operators.
- **DMBS independence** The implementation should work with any database supported by the Hibernate framework. This would be achieved by using the SQL standard both for representation and querying. The most complex task is the querying. Usually, the implementation of the fuzzy comparison operators is done in a procedural way. This is not possible in SQL. Therefore, it is necessary to use the SQL structure *CASE* $\sim$ *WHEN*.
- **Object-Oriented** The interface for the applications should be object-oriented. Both for the representation and the querying.
- **Performance** This property is essential for the acceptance of the system. If an application does not use the fuzzy representation or querying, the performance should be the same as with the version without the stratified implementation.

The proposed design model allows application developers to abstract the storage in an application from the running database. E.g., it is possible to develop an application within MySQL whereas the production environment is working with Oracle.

6.2.3. Implementation Model

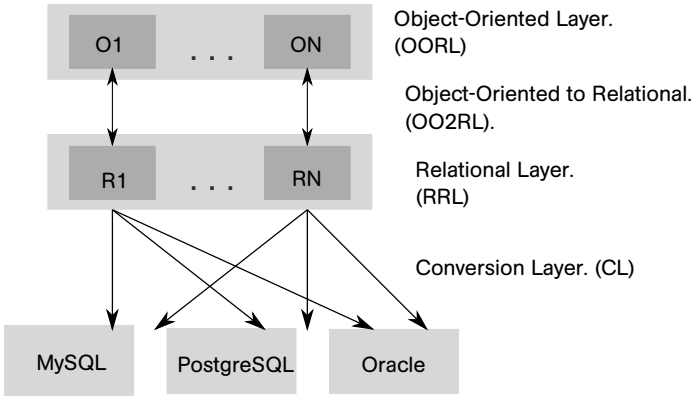


Figure 6.6: Abstract layer model

The architecture for a generalized object-oriented model needs the following elements (fig. 6.6):

- **Object-Oriented Representation Layer** *OORL*: The representation of the (fuzzy) types in an object-oriented way.
- **Object-Oriented to Relational Representation Layer** *OO2RRL*: This layer is a mapping between the objects in the higher layer and the representation in the layer below. Note that the mapping may not be trivial and a conversion function should be given.
- **Relational Representation Layer** *RRL*: The (fuzzy) types are represented by basic types. In the GEFRED model, FIRST is the specification for this layer.
- **Conversion Layer** *CL*: This layer customizes the SQL representation from *RRL* for the concrete database implementation. This customization process must be done because of the different implementations of the SQL standard on each DBMS.

6.2.3.1. Implementation

The proposed general framework relies between the application and the DBMS. The implementation of the model in the Hibernate Framework is the following:

- **Representation layer**: Is the implementation for OORL: the representation for the fuzzy objects in the Java programming language.
- **Adaptation layer**: This layer bring together the implementation of the OO2RRL and RRL layers.
- **Conversion Layer**: Hibernate supports this layer by means of the dialects. It has mapping types between SQL and the concrete implementation for these types in the database. Thus, each DBMS, has its own dialect. Hibernate provides dialects for the major DBMS, and it is easy to develop new dialects for new DBMS.

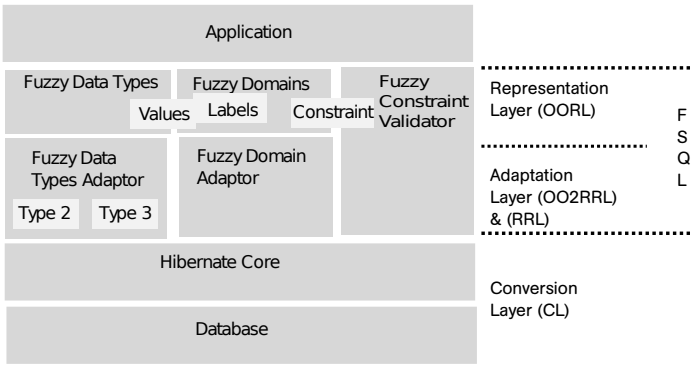


Figure 6.7: Detailed Hibernate architecture for fuzzy representation

In a more detailed view (fig. 6.7), the decomposition of the *representation layer* is depicted:

1. **Representation**:

   a) **Fuzzy data types**: Three fuzzy data types are represented (The representation for type 2 and type 3 fuzzy data types is shown in Tables 6.2 and 6.3 respectively). This representation is based on the Fuzzy Knowledge Representation Ontology (*FKRO*) [252] since it suits our object-oriented representation to a large extent.

   b) **Fuzzy domains**: To create fuzzy domains of types 2 and 3, two main fuzzy meta-domains are defined.

   c) **Fuzzy constraint validator**: Each fuzzy domain may be associated with a set of fuzzy constraints. The validator checks the constraints.

2. **Adaptation layer**: This layer transforms on the object-oriented representation to relational representation in the database. Thus, a fuzzy type over an ordered underlying domain is represented as an object in the upper layer. The database represents a fuzzy type over an ordered domain as five columns:

   a) **Fuzzy type**: A number indicating the fuzzy type stored in the other four columns. For a fuzzy domains of type 2, there are the following fuzzy types: *UNDEFINED*, *UNKNOWN*, *NULL*, *CRISP*, *INTERVAL*, *APPROX*, *TRAPEZOID*.

   b) **Fuzzy values**: Fuzzy domains of type 2 are stored in four columns. For example, if the fuzzy number stores a trapezoid, then each point is represented in a column in the database. In Table 6.2 there is a short description for each fuzzy data type.

   Fuzzy domains of type 3 have a variable length representation. The basic representation needs at least 3 columns:

   a) **Fuzzy type**: a number indicating the fuzzy type stored in the other four columns. There are the following fuzzy types: *UNDEFINED*, *UNKNOWN*, *NULL*, *SIMPLE*, *POSSIBILITY DISTRIBUTION*.

   b) **Fuzzy values**: Pairs of values: Label ID and possibility degree. There is a short description in Table 6.3.

   Thus, the adaptation layer has to deal with the representation of fuzzy domains of types 2 and 3. Note that fuzzy domain of type 1 is represented by a numeric Hibernate basic type, therefore it is not necessary to adapt the representation.

6.2.4. Valid time representation

As mentioned in Chapter 3, in Section 3.2.1 there are different approaches for dealing with imperfections in valid-time. In this work, we have implemented the two main proposals. First of all, the implementation of a Fuzzy Validity Period, FVP is explained. Then, a second approach based on the Possibilistic Valid-time Period, PVP is also discussed.

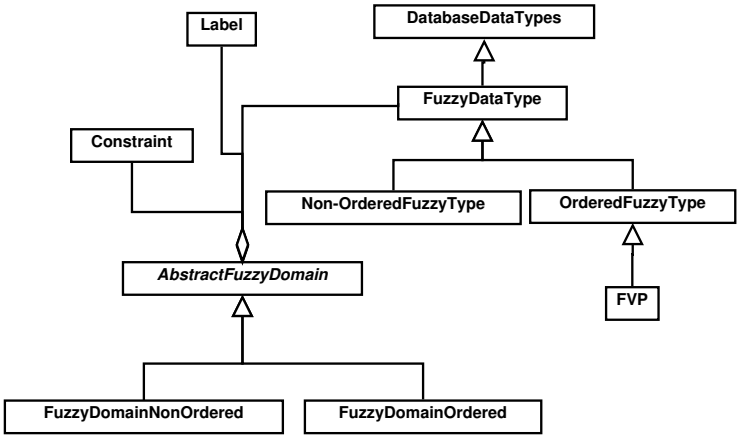


Figure 6.8: UML diagram for fuzzy data types.

Fuzzy Type	FT	F1	F2	F3	F4
UNKNOWN	0	N	N	N	N
UNDEFINED	1	N	N	N	N
NULL	2	N	N	N	N
CRISP	3	d	N	N	N
LABEL	4	ID	N	N	N
INTERVAL	5	n	N	N	m
APPROX	6	d	d-m	d-m	m
TRAPEZ	7	α	β	γ	δ

Table 6.2: Relational representation for fuzzy attributes type 2. Note that N is the abbreviation for the *NULL* constant.

Fuzzy Type	FT	FP1	F1	...	FPn	Fn
UNKNOWN	0	N	N	...	N	N
UNDEFINED	1	N	N	...	N	N
NULL	2	N	N	...	N	N
SIMPLE	3	p	d	...	N	N
POSS.DIST	4	p_1	d_1	...	p_n	d_n

Table 6.3: Relational representation for fuzzy attributes type 3. Note that N is the abbreviation for the *NULL* constant.

Fuzzy Type	FT	F1	F2	F3	F4
UNKNOWN	0	N	N	N	N
UNDEFINED	1	N	N	N	N
NULL	2	N	N	N	N
TRAPEZ	7	α	β	γ	δ

Table 6.4: Relational representation for fuzzy validity period, a fuzzy domain of type 2 attribute. Note that N is the abbreviation for the *NULL* constant.

Description	D_s	a_s	b_s	D_e	a_e	b_e
An ill-known time interval	V	V	V	V	V	V
An unknown time interval	N	N	N	N	N	N
A left-open time interval	N	N	N	V	V	V
A right-open time interval	V	V	V	N	N	N

Table 6.5: Relational representation for a possibilistic valid-time period in the form $[S, E]$ with $S = [D_s, a_s, b_s]$ and $E = [D_e, a_e, b_e]$. The value *N* is the notation for the *NULL* constant. The value *V* is the notation for a time value.

6.2.4.1. Fuzzy Validity Period

The Fuzzy Validity Period FVP is represented in the framework as mentioned in Section 2.1. The underlying ordered domain is the Julian Day Number (JDN). Thus, the representation of this data type on the framework is based on a fuzzy data type with an underlying ordered domain.

A fuzzy underlying domain is defined with some operations to convert between Java dates (usually `java.util.Date`, in Gregorian calendar format) and the Julian Day Number with the formula explained in Section 2.1 and the algorithm in [93].

Table 6.4 is the relational representation of the FVP in a database. Five columns are needed: The first one (Fuzzy Type, FT) stores the subtype for the object. Four values are allowed in order to represent a fuzzy validity period: the constants *UNKNOWN, UNDEFINED* and *NULL* and the trapezoidal possibility distribution. The following four columns (from F1 to F4) store the values for a given element.

6.2.4.2. Possibilistic Valid-time Period

A possibilistic valid-time period is a more convenient representation for a valid-time interval, as pointed out in Section 2.1. In this case, in order to simplify the querying, the dates are stored as a *long* data type. In order to store a PVP, two possibility distributions are stored, one for the start point and one for the end point. In the implementation, only triangular membership functions are considered. The corresponding relational representation is shown in Table 6.5.

In the following, the usage of the two types will be illustrated with an example.

ID	VID	Entity	FVP
3	001	King Henri	[1203, 1205, 1206, 1207]
4	001	King Phillipe	[1210, 1215, 1216, 1217]
5	001	Pope Alexander I	[1222, 1223, 1226, 1228]
3	002	King Henri	[1253, 1255, 1256, 1257]

Table 6.6: Historical documents database. Representation of valid-time by a FVP. In order to simply the representation, only the year is shown.

Example 38. Consider a database with diplomatic documents. In this case, the documents managed are from the Medieval age. For each document, an identifier (ID) is stored, the name of the legal entity and the approximate starting and ending dates in which the document was valid. We will illustrate how to define the entity classes for using either the PVP or the FVP. Table 6.6 shows the version of the FVP.

The definition for an entity class that models a historical document is the following:

```
 /**
 * Entity class representing a diplomatic document.
 */
@Entity
@Table(name = "document", catalog = "")
public class Document implements Serializable {

/** Primary key, with two fields:
    an identifier, ID and a version VID.  */
@EmbeddedId
@Type(type = "es.jpons.
temporal.types.TemporalPKType")
@Columns(columns={
 @Column(name="id"),
 @Column(name="vid")
    })
private TemporalPK tid;

/**  Field Legal entity  */

@Basic(optional = false)
@Column(name = "entity")
private String entity;
/** Fuzzy Validity Period */
@Type(type = "es.ugr.decsai.
fsql.databasedata types.OrderedAFTUserType")
@Columns(columns = {
 @Column(name = "FVP1"),
 @Column(name = "FVP2"),
 @Column(name = "FVP3"),
 @Column(name = "FVP4"),
 @Column(name = "FVPT")
 })
```

ID	VID	Entity	PVP					
			D_s	a_s	b_s	D_e	a_e	b_e
3	001	King Henri	[1203,	5,	5]	[1207,	1,	1]
4	001	King Phillipe	[1215,	5,	5]	[1216,	1,	2]
5	001	Pope Alexander I	[1223,	1,	2]	[1226,	2,	2]
3	002	King Henri	[1255,	3,	3]	[1257,	4,	4]

Table 6.7: Historical documents database. Representation of valid-time by a PVP. In order to simplify the representation, only the year is shown.

```
34 private FVP fvp;
```

In order to create a new document that represents the first row in Table 6.6 the code is the following:

```
Document d = new Document(new TemporalPK(3,1),
2           "King Henry",
            new FVP(1203, 1205,1206,1207));
```

The representation for the PVP case is shown in Table 6.7. The definition of the class with a possibilistic valid-time period is exactly the same, but now, the possibilistic valid-time period PVP is declared in the following way:

```
1 /** Valid-time  */
      @Embedded
3     @Columns(columns={
          @Column(name="startMP"),
5         @Column(name="startR"),
          @Column(name="startL"),
7         @Column(name="endMP"),
          @Column(name="endR"),
9         @Column(name="endL")
      })
11    protected PossibilisticVTP pvp;
```

The code to create an instance of a document is the following. The document has the ID = 3 and the VID = 001 and was written by the king Henry (first row in Table 6.7).

```
1 Document d1 = new Document(new TemporalPK(3,1),
                "King Henry");
3 PossibilisticVTP pvp =
          new PossibilisticVTP(1203,5,5,1207,1,1);
5 d1.setPvp(pvp);
```

6.2.5. Fuzzy Querying

In this part, two different approaches are presented. The first approach illustrates how to implement the fuzzy comparison operators by SQL statements. The second approach presents a different querying strategy by using ill-known constraints.

6.2.5.1. Approach 1: Fuzzy operators in plain SQL

To generalize fuzzy querying into any relational database, we should take into account that the interface between the relational DBMS and the framework is SQL standard. Therefore, the model we propose uses the following elements:

- **Declarative implementation** for each fuzzy operator. This implementation should be done in the SQL language.
- **Abstract syntax tree (AST) representation** for the query. These representation allows a customization process done by the conversion layer (*CL*).
- **Conversion Layer**: This layer customizes the *AST* for the running database.

The implementation of fuzzy querying is done by modifying the HQL language. The fuzzy operators are implemented in a declarative way in the SQL language. The HQL language has the following features:

- **Object-oriented representation** for queries.
- **Customization**: The HQL code is analyzed and translated into SQL statements. Hibernate customizes SQL statements for the running DBMS through the dialect. E.g. if the application is running against MySQL, then Hibernate generates the SQL code customized for it.

The Hibernate core deals with the HQL translation. By modifying this code, implementing fuzzy operators should result in a HQL with fuzzy querying capabilities. This extension will work on any database supported by the system also. The following s are how Hibernate processes a HQL query:

- The framework builds an abstract syntax tree (AST) once the query passed lexical and syntactical analysis.
- The AST represents tokens as nodes. The semantic analyzer renders the tree into SQL statements. Then the dialect customizes the SQL statements.

The following example explains the translation workflow.

Example 39. Consider a restaurant database including data about restaurants. For each restaurant, the following information is stored: A unique identifier, ID, the name of the restaurant and the average price. This is stored as a fuzzy type 2, by using a triangular membership function. Consider the following query:

"The user wants to obtain a list of restaurants with an average price around 15 euro".

This is translated to the following HQL sentence, using the fuzzy equals operator:

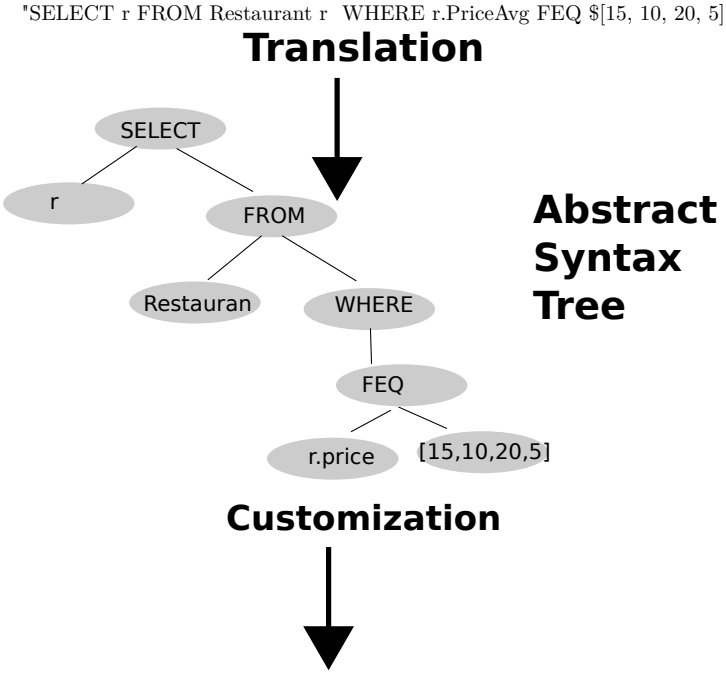
```
1 SELECT r FROM Restaurant r
  WHERE r.PriceAvg FEQ $[15, 10, 20, 5];
```

Figure 6.9 shows the AST tree for this sentence.

Then, the FEQ node is mutated in the rendering process to its implementation in SQL. Through the dialect, the sentence is customized to execute for instance, in MySQL.

HQL sentence:

"SELECT r FROM Restaurant r WHERE r.PriceAvg FEQ $[15, 10, 20, 5];"



SQL sentence:

"SELECT * FROM Restaurant as r
WHERE
1 < CASE WHEN (r.priceAvg.gamma <= beta2)
OR (r.priceAvg.beta >= gamma2)
THEN 0
WHEN (r.priceAvg.alpha = alpha2) THEN 1
WHEN (r.priceAvg.gamma > beta2) AND (r.priceAvg.alpha < alpha2)
THEN (r.priceAvg.gamma - beta2) / (r.priceAvg.delta - delta2)
ELSE (gamma2 - r.priceAvg.beta) / (r.priceAvg.delta + delta2);"

Figure 6.9: Translation from HQL to customized SQL statements. From the left to the right, the HQL query is translated into an AST. The dialect customizes the AST for the running database into specific SQL statements.

```
SELECT * FROM Restaurant as r
WHERE
1 < CASE WHEN (r.priceAvg.gamma <= beta2)
OR (r.priceAvg.beta >= gamma2) THEN 0
WHEN (r.priceAvg.alpha = alpha2) THEN 1
WHEN (r.priceAvg.gamma > beta2)
AND (r.priceAvg.alpha < alpha2)
THEN (r.priceAvg.gamma - beta2) /
( r.priceAvg.delta - delta2 )
ELSE (gamma2 - r.priceAvg.beta) /
( r.priceAvg.delta + delta2 );
```

The following example illustrates the use of temporal operators.

ID	Name	Salary	BossID	Cat.	FVP
001	Josh	1200	002	C	[10/10/2009, 27/10/2009, 19/10/2010, 27/10/2010]
001	Josh	1500	002	B	[19/10/2010, 27/10/2010, -, -]
002	Robert	800	005	A	[14/05/2007, 25/05/2007, 17/01/2008, 30/01/2008]
002	Robert	1200	005	A	[17/01/2008, 30/01/2008, 24/05/2009, 30/05/2009]
002	Robert	1400	003	A	[24/05/2009, 30/05/2009, 28/10/2010, 30/10/2010]
003	Alex	2100	-	A+	[02/05/2007, 15/05/2007, -, -]
004	Tyna	1300	002	A+	[25/07/2009, 30/7/2009, 15/10/2009, 25/10/2009]
005	Rose	2300	003	A+	[25/09/2010, 30/09/2010, 25/02/2011, 30/02/2011]

Table 6.8: The relation employees with fuzzy validity periods (FVP).

Example 40. Consider a company which stores data about its employees. The data is stored in a fuzzy valid-time database (see Table 6.8). Each time the relation is updated, a new row with an updated version of the data is stored. The starting and the ending points of the validity period are not precisely known, and are represented by an FVP given in the $[\alpha, \beta, \gamma, \delta]$ format explained in Section 3.2. For simplicity, the values for each element of FVP are shown in their corresponding Gregorian calendar but are stored in the JDN format mentioned in Section 2.1.

Class definition An entity class represents a table in the database. A field of an entity class represents one or several table columns. The corresponding class declaration for the Table 6.8 is:

```
public class Employee implements Serializable {

private String ID; // primary key: ID
private String name; // name of the employee
private Double salary; // salary
private Employee boss; // boss
private Category category; // category
private FVP fvp; // fuzzy validity period
}
```

Querying Consider the user has the following query:
"Find all the employees with boss 002 during the same period of time."
The translation of this query into HQL is the following:

```
SELECT e,f
FROM Employee e, Employee f
WHERE e.ID="002" AND e.ID<>f.ID
AND e.fvp EQUALS f.fvp;
```

The framework translates the HQL query to SQL statements as explained in section 6.2.5. Then, the statements are sent to the database. The resultset of the query returned by the database is mapped backwards to objects by Hibernate. Table 6.9 shows the resultset for the query, and the compatibility for the result, computed by the fuzzy equals operator, *FEQ* (see Example 39 and Section 2.4.1).

e.ID	e.name	f.ID	f.name	Comp.
001	Josh	004	Tyna	$0,55$

Table 6.9: The relation employees with fuzzy validity periods (FVP).

The following subsection is devoted to present the second querying approach.

6.2.5.2. Approach 2: Querying by ill-known constraints

This approach presents a simple querying architecture, which is illustrated in figure 6.10. In this case, when the user makes a query, a minimal subset of the result is fetched to the Hibernate layer. Then, the global membership degree is computed by using ill-known constraints (See Section 3.1.4). In this case, the implementation has been done using the Criteria API.

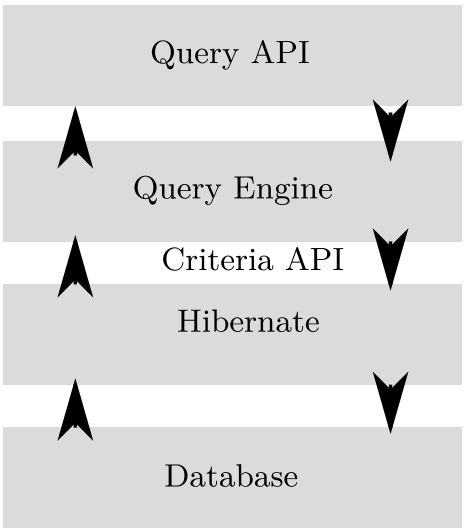


Figure 6.10: Architecture for the query engine. The top layer exposes the querying API, which is used directly by the applications. The query engine uses the criteria API and other Hibernate functionality. On the bottom of the architecture is the relational database supported by the framework.

The elements of the architecture are explained in detail:

- Query API: A set of querying operations are provided to the application developer. The Allen operators are provided to compare two possibilistic valid-time intervals PVP.

- Query Engine: The query engine performs the operations exposed in the Query API and returns a resultset that fulfills the operation. The workflow for the query engine is the following:

  1. An operation in the Query API is called within the application.
  2. The query engine translates the query into a set of Criteria.
  3. The set of Criteria is sent to the Hibernate Framework and translated into SQL statements which are executed in the underlying database.
  4. If the query returns a resultset, it is wrapped into its corresponding entity class by Hibernate.
  5. The resultset is returned to the query engine. The resultset returned to the query engine fulfills with a degree greater than zero the operation. Now the possibility degree is computed for each tuple in the resultset (See Section 3.1.4) . Finally, the resultset with the corresponding possibility degree is returned to the application.

Example 41. Consider the employees database in the previous example. Now, the valid-time is stored using a possibilistic valid-time period, PVP as presented in Table 6.10.

ID	VID	Name	PVP	
001	001	Josh	[18/10/2009, 8, 8]	[18/10/2010, 8, 8]
001	002	Josh	[19/10/2010, 8, 8]	UC
002	001	Robert	[20/05/2007, 5, 5]	[20/01/2008, 3, 10]
002	002	Robert	[21/01/2008, 3, 10]	[25/05/2009, 1, 5]
002	003	Robert	[26/05/2009, 1, 5]	[25/10/2010, 5, 5]
003	001	Alex	[15/05/2007, 12, 2]	UC
004	001	Tyna	[26/07/2009, 1, 5]	[16/10/2009, 2, 9]
005	001	Rose	[27/09/2010, 2, 3]	[27/02/2011, 2, 3]

Table 6.10: The relation employees with possibilistic valid-time periods (PVP). For simplicity, the dates are expressed in *dd/mm/yyyy* format. Other attributes like *Salary*, *BossID* and category are omitted, but have the same values as in Table 6.8.

In this case, the user has the following query:

"Find all the employees working for the company during the period around the middle of October 2009 to the middle of October 2010 and whose salary is between 1.000 and 1.500 euros."

The query has to be translated into the following code:

```
Employee emp = new Employee();
PossibilisticVTP pvp =
        new PossibilisticVTP(
         15,10,2009, // day month year
         5, 5,  // days in the left and right margins
         15,10,2010,  // day month year
         5, 5 ); // days in the left and right margins

```

```
// create the temporal query for the employee entity:
 TemporalQuery tq =
        manager.createTemporalQuery(emp);
// set the allen relation and the pvp:
tq.setAllenRelation(AllenRelation.during, pvp);
// obtaining the resultset:
List<QueryResult> list = tq.getList();
```

The resultset obtained is shown in Table 6.11.

global	nts	ts	**ID**	**VID**
0,6923	1	0,38461	001	001
0,5	1	0	001	002
0,5	1	0	002	002
0,5	1	0	002	003
0,5	1	0	004	001

Table 6.11: Resultset table for the query. In this table, the value *global* is the aggregation of both temporal and non-temporal satisfaction. The values for the temporal (column *ts*) and non-temporal (column *nts*) satisfaction degrees are shown. Furthermore, only the values for the primary key are presented.

In the following, we will study whether despite of the payload of the Criteria API the second approach is more efficient.

6.2.5.3. Comparison

In this subsection we will compare the two presented querying approaches. We are going to compare the execution time of the Allen relation *Before* with both approaches. The configuration of the testing environment is the following:

- The operating system is Ubuntu 10.04. Kernel version 2.6.32-22.
- The database for the test is MySQL 5.1.63-0ubuntu0.10.04.1 (Ubuntu).
- The server has the following configuration:
  - Processor: Intel i5 CPU M 520@2,40Ghz.
  - RAM: 6GB DDR3.
  - HD: 2 x 500 GB

A single table will be used for the test. The employee table. The tests will be done against the same table but with increasing number of rows. The rows will be generated synthetically and are identical for the two approaches. A sample of the employee table is shown in Table 6.10. The execution time for each test is the average of ten executions of each approach to the same dataset. The results are plotted in a graph. Figure 6.11 shows the performance of each approach. It is shown how the first approach is faster on smaller datasets, whereas the second approach is faster on large datasets. The reasoning behind that behaviour is the following: The two step approach

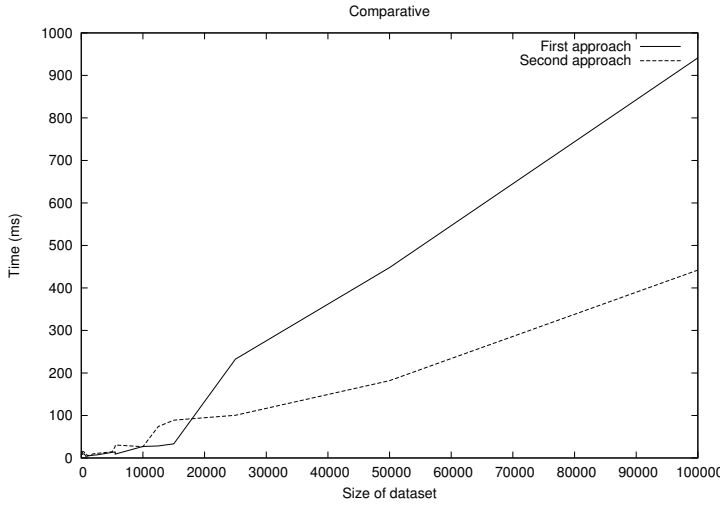


Figure 6.11: Comparative for the execution times between the two proposed approaches.

is slower on small datasets just because the size of the dataset. For bigger datasets, the execution time of the first approach is slower than sending a very optimized query to the database and process a small dataset in the server side, to obtain a satisfaction degree. It is important to explain that we have tried to minimize the optimization effects that the cache provides, to get a worst-case execution. Due to the fact that some parts of the cache system can not be disabled, the high variations in the execution time when working with the smallest datasets is because of the cache system.

6.3. Related work

In this section we will discuss the differences with respect to the portability among several implementations. The proposals analyzed are:

- **FSQL server** by [54]: The reference implementation of the FIRST interface on the GEFRED model. The first implementation works with an Oracle database, although there is an implementation in PostgreSQL. A new module called Fuzzy Valid Time Support Module (FVTM) has been developed to implement the theoretical model in chapter 3.
- **SQLfi** by [56]: The implementation for the SQLf language [201].
- **FDBLL** by [253]: Fuzzy database language and library. A fuzzy SQL implementation in C language over a relational DBMS.
- **PSQL** by [254]: An extension of the FSQL model. The main features are the use of priority fuzzy logic and the portability: The implementation is done by a JDBC driver. The driver acts as interface between any Java program and the fuzzy database. The user may change the running database by just adding the fuzzy meta tables to the catalog and keeping the same program.

Fuzzy DB	Catalog	Interface	Query language	Query Processor
FSQL	Inside DB.	FSQL client	FSQL	Procedural.
SQLfi	Inside DB.	Client app.	SQLf	Procedural.
FDBLL	Inside DB.	Client app.	Fuzzy SQL	Procedural.
PFSQL	Inside DB.	JDBC client	PFSQL	Procedural.
H. FSQL	Outside DB.	Entity Classes	Fuzzy HQL	Declarative.

Table 6.12: Comparison among different fuzzy DB implementations.

Fuzzy DB	Changes for representation	Changes for querying
FSQL	Create the metadata tables (FMB). Develop a client application.	Implement in a procedural way the FSQL operators. Develop a client application.
SQLfi	Create the metadata tables. Develop a client application.	Implement the fuzzy operators and the query translator.
FDBLL	Create the fuzzy data definitions.	Implement the fuzzy SQL processor in the DBMS.
PFSQL	Create the metadata tables.	No changes
H. FSQL	No changes.	No changes.

Table 6.13: Changes to migrate the implementation to another DBMS.

- **Hibernate FSQL**: The proposed implementation. The main difference is that the fuzzy meta data are not stored in the database, therefore, changing the running database is as easy as changing some parameters in the Hibernate configuration file. There is no need to create or modify fuzzy meta tables in the DBMS catalog.

Table 6.12 shows the main differences in the implementations and in the portability among each approach. Table 6.13 shows the changes that must be done in order to change the running DBMS for an application.

6.4. Conclusions

In the presented implementation we have introduced a general model for the representation of (fuzzy) types and for fuzzy querying. The main advantage with respect to other implementations is portability. The drawback for the portability is the dependency between the application and the framework. This means that, outside the framework, the DBMS is not able to manage the fuzzy types nor to make fuzzy queries. This is not such a big issue. Over the last few years the trend is to develop the business layer outside the DBMS too.

The temporal extension allows both representation and querying by means of fuzzy temporal operators. The main contribution allows to represent, handle and query time-variant objects. It is possible to represent imperfect time intervals for the validity period of a given object.

The novel contributions of this chapter are:

- The implementation within the Hibernate framework of the theoretical model proposed for valid-time databases.
- The implementation of the temporal flexible querying operators by using a relational implementation in SQL an a procedural implementation by using the criteria API.

7 Conclusions and further research work

This chapter summarizes the main contribution of this thesis. We will explain how the research objectives introduced in Chapter 1 have been resolved. Next to that, the main contributions are discussed. Finally, challenges and new research opportunities are studied.

7.1. Summary

This thesis is concerned with the treatment of uncertain or incomplete temporal data in an information system. In order to accomplish this objective, the following four theoretical tasks have been done:

1. First a mathematical framework to represent and deal with the relationships of the time intervals is proposed.
2. A formal model to represent and handle time-dependent entities in a relational database is implemented.
3. An extension to the bipolar querying allows to make complex queries with temporal constraints.
4. An extension based on the Triangular Model deals with the visualization of uncertain time intervals.

These theoretical proposals have been implemented into open software prototypes and applied in to real-world applications. The following development tasks have been done:

1. The representation of possibilistic temporal intervals as well as fuzzy data types have been defined and implemented within the Hibernate framework.

2. The querying of the previously defined data types is done by using the implementation of the operators defined in the theoretical model. For fuzzy data types, fuzzy operators are implemented. For the possibilistic temporal operator, the possibilistic version of the Allen's operator are implemented. In order to improve the performance, two sub-tasks have been done:

   a) A declarative implementation of the comparison operators.

   b) A procedural and more efficient implementation of the comparison operators.

3. In order to keep the database consistency the consistence mechanism proposed by the theoretical model have been implemented.

In the following, we will summarize the contributions of this thesis with respect to the research objectives described in Chapter 1.

Objectives

1. Definition and formalization of temporal data types and operators. The goal is to abstract the main characteristics of the temporal data types. Then, it would be possible to re-define the data models and the operations in a database.

   Time has been shown to be a complex concept. First of all, it has been studied how humans beings handle time indications. The main conclusion is that humans deal with uncertain temporal expressions in their daily life. Therefore, in order to achieve a realistic implementation of treatment of temporal information, a formal tool to deal with the uncertainty, imprecision and or imperfection in the temporal expressions is proposed. The three main theoretical frameworks to deal with imperfections in time are rough sets, probability theory and possibility theory. Among them, we choose possibility theory for its intrinsic capability of dealing with uncertainty in knowledge. Hence, a proposal in the framework of possibility theory has been done to represent imperfect temporal information by using the possibilistic valid-time intervals PVP. The relationships between these intervals have been modelled by using the ill-known constraint framework proposed.

   As part of this main objective we find the following objectives.

   a) To define a representation for the temporal elements. The chosen representation should allow uncertainty and vagueness within the temporal elements.

   In order to define a temporal representation we studied the proposals that can be found in the literature. There are several frameworks to deal with vagueness and uncertainty. Nevertheless, we find these proposals to be incomplete. After some research, we concluded that the best way for the representation of temporal elements is the time intervals. To handle uncertainty and / or vagueness, the best solution is to use the possibility theory to model such imperfections. Time intervals have both starting and ending points. Typically, the imperfections in the modelling of these time intervals

are related to the modelling of imperfection of one or both points. Therefore, in our proposal, two possibility distribution describe a time interval. It is possible to represent crisp time intervals as well.

In the other hand, the representation of time intervals is provided with a visualization method. In this thesis we have extended the triangular model, used in the representation of crisp and rough time intervals to uncertain time intervals. The visualization in the triangular model makes easier to compare and classify big amounts of temporal intervals.

b) To define the operations and semantics for the possible relationships between the temporal elements defined above.

The relationships between time intervals have been studied in depth. First Allen studied all the possible relationships between two crisp time intervals. Then, several authors provided a fuzzy version of these relationships. In this thesis we propose a general framework (based on the so called ill-known constraints)for the possibilistic evaluation of sets. As a particular case applied on time intervals, we define the possibilistic counterpart of the Allen's relations. It is possible to define even more complex relationships and operations. For example, the operator *Close* that closes an open time interval with respect to another time interval. This operator is defined by using several ill-known constraints.

c) To extend the fuzzy relational model to represent the temporal elements and to support the operations on them.

There are plenty of crisp temporal database models. Each model fits a need. There are three main types of temporal databases: transaction-time databases, that are used on accounting systems, valid-time databases that model time-dependent objects. Decision-time databases model the time when a decision about some fact was done. It is possible to combine these three types of time into a multi-temporal database, but since transaction and decision-time are timestamps, the only time which is subject to imperfection is valid-time. Therefore, we applied the representation and the operators defined in the previous objective to implement a possibilistic valid-time model in a relational database.

By using the bipolar querying, it is possible to give more expressive power to the user. The temporal querying extension is build on the top of the satisfaction - dissatisfaction approach. Several proposals with respect to the temporal constraint have been done and some ranking and classification method have been provided. As a result of the query, the user obtains a bipolar satisfaction degree BSD. This value classifies the results within three groups. The first group are the results which do not satisfy at all the criteria in the query. The second group is the set of results that are indifferent to the user. That is it, a set of results that satisfy some criteria and dissatisfy some others. Finally the third group of results, is a set of results that fulfil the positive and negative criteria provide by the user. As it has been shown, this approach gives some extra information when querying at the same time that provides extra expressive power.

d) To define the behaviour and the semantics for the time-dependent data. In order to allow imprecision, the theoretical model is build on the top of the GEFRED model. The proposal deals with the two main issues of temporal database models: the problem of the primary key and the consistence problem. With respect to the primary key, a version number is provided to support several versions of the same object with different valid-time values. The consistence mechanism is provided by re-defining the data manipulation language (DML) of the model.

2. Implementation of the theoretical model obtained in the previous stage.

   The implementation of the possibilistic model for temporal databases have been developed and implemented in an open-software framework called Hibernate. That framework is an object-relational mapping tool that provides extra querying capabilities. We have chosen this framework because of it is not database dependent and because it is open source. The framework has defined different dialects for each database. For example MySQL has a specific dialect, Oracle has also a different dialect, etcetera. It is possible to change the underlying database, by just changing the dialect. Each dialect customizes the SQL code for the specific database. In this thesis we have extended that framework to support the fuzzy data types defined in GEFRED as well as the fuzzy operators to compare them. On the top of this, we build the temporal data types as well as the comparison of time intervals. The implementation provides two different approaches of the model. The first approach provides the implementation of the fuzzy operators by using SQL sentences. In other words, it provides a declarative implementation for the fuzzy operators. The main benefit of this is that the full query processing is done in the database. When the query is complex, the SQL text that is sent to the database is also complex and difficult to understand by a human being. The second approach, makes an procedural implementation of the fuzzy operators. In this case, the full query processing is done at the level of the Hibernate layer, not in the database. As we studied for big datasets it has been shown to be more efficient than the first approach.

   As an extension of the implementation work developed on this thesis, an implementetion has been proposed on the top of the Oracle database. The fuzzy valid-time support module is a module that extends the functionality of the previously developed Fuzzy Object-Relational Database System. In this case, the implementation has a different approach with respect to the presented model. Some of the data manipulation language DML sentences have been redefined. In this case, the approach is closer to the implementation provided by the Oracle Workspace Manager, but using the logic of the ill-known constraint presented in this thesis.

7.2. Discussion

This section studies the main contributions of this thesis and provides a discussion for further research work. First, the value of this research is explained and then the

remaining challenges and future research lines are proposed.

7.2.1. Representation of time

In order to properly represent the time in an Information System, some simplifications should be done. First of all, time is a continuous magnitude, due to the internal representation in a computer, the time has to be discretized. This first step introduces some amount of uncertainty as explained. The smallest unit of time that can be represented in a system is called *chronon*.

The second step when modelling time is to decide whether to represent the time as points or as intervals. It has been shown that both approaches are equivalent, since a time interval can be modelled by a set of time points and vice versa, a time point is equivalent to a time interval in which both starting and ending points are the same time value. Most of the systems have a time interval representation for time values.

As explained in Chapter 2, several studies demonstrate that humans beings inherently manage the time with imprecision and / or uncertainty. Therefore, several proposals have been done in order to deal with this. Nevertheless, there was not a unified approach to represent and handle with time intervals. In Chapter 3, we provide a formal framework to deal with time intervals (called ill-known time intervals) which may contain imprecision or uncertainty in one or both points of the interval. In an ill-known time interval, two possibility distributions are defined. One for the starting point and another for the ending point. By doing this, we provide a way to compute both possibility and necessity measures for every calculation within the time interval. The main advantages of this is that both measures can be used in the ranking or in the classification of the results.

The visualization of time intervals is also presented in this thesis. In chapter 3 we have shown the representation in a two dimensional axis of possibilistic valid-time periods (PVP). In the x-axis, is represented the time and in the y-axis is represented the possibility degree. This representation is enough for small quantities of time intervals. When the number of intervals increase, the graphical representation becomes messy. The triangular model, represents crisp time intervals in a two-dimensional space. A time interval represented in a one dimensional scale is therefore represented as a point in a two dimensional scale. In Chapter 5 we propose an extension of the triangular model to represent uncertain time intervals. By using this visualization, the study of the relationships between time intervals becomes clearer.

7.2.2. Temporal relationships

The relationship between time intervals have been studied in depth. In 1983, Allen studied all the possible relationships between two crisp time intervals. As a result the thirteen Allen's relations were obtained. As detailed in Chapter 2, there are several proposals to extend the Allen's relations to the fuzzy case. In this thesis, we provide a formal tool (the ill-known constraints) to model and specify the relationships between ill-known time intervals. By using the ill-known constraints not only the Allen's relations can be modelled but also some more complex relationships.

In order to visualize the temporal relationships the triangular model is used. Each

one of the Allen's relations has a specific area in this representation. Hence, with a simple visual inspection of a time interval represented in this model, the corresponding Allen relation is obtained. In this thesis we propose the extension of the triangular model to visualize uncertain time intervals. As result, in the surrounding areas of the uncertain time interval, there is also uncertainty about the Allen relation that actually applies. We defined these areas (for example: possibly meets, possibly overlaps) and provided a method to compute the possibility measure of the Allen's relations.

7.2.3. Time in databases

Time in databases has been studied for a long time. There are three different types of time that are handled specifically by a temporal database. Transaction time is a time stamp that establishes the time when a fact was recorded in a database. It must be in the past. In the other hand, valid-time specifies the time when a fact is true in the modelled reality. In this case, valid-time may have values in the past, the present and / or the future. Usually valid-time is represented as a time interval. Finally, decision-time is also a time stamp which sets the time when a decision about the fact recorded in the database was done.

Among these three types of time, the valid-time could be affected by imperfections such as imprecision, vagueness and / or uncertainty. Therefore, in the model proposed in Chapter 3 we only deal with valid-time.

In a temporal database, several issues have to be addressed, as explained in Chapter 2. The primary key has to be re-defined to allow several versions of the same object. But also a consistence mechanism has to be provided in order to ensure that no spurious values are inserted in the database. This is achieved by re-defining the data manipulation language DML. For example, before inserting a new version of an existing value, it is checked if the new version overlaps with some previous versions. If there is some amount of overlapping, then the insertion is rejected.

In Chapter 3 we re-defined the DML sentences provided by the GEFRED model to provide the consistence mechanism explained before.

7.2.4. Bipolar querying of temporal databases

Bipolar querying of databses provides a more powerful way to model user's preferences. There are two main frameworks as explained in Chapter 4. It has been shown that both frameworks are equivalent in terms of expressive power. In other words, the same query can be expressed in both frameworks.

In this thesis, we have extended the satisfaction-dissatisfaction approach to deal with temporal queries. First we provide a study of the temporal constraints in the bipolar query specification. We obtained that there are two main places to specify a temporal constraint. At a global level, we want the results obtained to be within a temporal frame specified by means of an Allen relation and a ill-known time interval. It is possible to specify a temporal constraint within the elementary query conditions. Then, for each specific constraint we could specify a temporal frame to be valid.

As it has been studied, the main problem when dealing with time in a bipolar query is the aggregation. From the evaluation non-temporal bipolar query we obtain a bipo-

lar satisfaction degree BSD. This degree is usually scaled in the interval $[-1, 1]$ and provides to the user valuable information. The classification provided by a BSD can be split into three groups. Results within $[-1, 0]$ are said to be the set of non-desirable results. That is, the set of results that do not fulfil the requirements of the query. Results near the value of 0 are said to be indifferent. Finally, results closer to 1, are said to be desirable results.

The main problem is to aggregate the BSD, with the evaluation of the temporal constraint. In this thesis we provide a method to order the results in the temporal constraint first and then a flexible aggregation method. This aggregation method allows to the user to specify a weight ω (typically between $[0, 1]$) for the temporal constraint with respect the non-temporal constraint. A default value of $\omega = 0.5$ has been shown to be balanced for most of the queries.

7.2.5. Implementations

Despite the efforts to add a sound temporal support to databases, the standards usually offer a little support for date / time dependent objects if any. The SQL standard defines several temporal types (DATE, DATETIME and TIMESTAMP) but, the implementation rely on the developer of the database. Because of that, each implementation of temporal data types in commercial systems have a different implementation, behaviour and operators. For example, Oracle handles each date type as a Julian Day Number (explained in Chapter 2), so it supports dates from the 1st of January of 4712 B.C. whereas MySQL supports dates from the 1st of January of 1001 A.C. The sematics of the operators is also database-dependent. For example, adding a value of 1 to a date in Oracle means adding one day to the date whereas in MySQL means adding one second.

Some efforts have been done to join the temporal management into the SQL standard. The language TSQL handles bi-temporal (both transaction and valid-time) databases. This language, with some modifications was proposed to join the SQL standard without success. As part of the research in temporal databases, some prototypes of temporal database implementations have been proposed, as discussed in Chapter 6. But, to the best of our knowledge there are a few commercial implementations that support some aspects of temporal databases. An example of that is the Workspace Manager by Oracle. This framework enables temporal support for non-temporal databases. But the aim of this implementation is not only the temporal support but speed up the workflow between the user and the database for very large datasets. The main idea is that the user work within a time frame and therefore, he or she works with a subset of the data. Hence, the framework provides several mechanisms to deal with conflicts when inserting, updating or deleting data.

In this thesis we provide an implementation on the top of a well-known open source framework, Hibernate. This framework is widely used for the object-relational mapping. At the same time, this framework acts as an abstraction layer between the current database system used by the application and the application itself. This abstraction is based in the so called dialects. Each database vendor has a specific dialect, and therefore by changing the dialect, it is possible to change the underlying database system.

Due to the abstraction that Hibernate does about the underlying database, it is pos-

sible to implement the theoretical model the possibilistic valid-time approach explained in Chapter 3. This implementation will work on the top of any of the databases supported by Hibernate. In other words, with only one implementation, is possible to represent time dependent objects with possibilistic valid-time periods and to query by using the temporal relations defined in the theoretical model.

The implementation provides two methods for the querying. The first method consist on a declarative implementation by using SQL sentences. The second implementation is a procedural implementation of the comparison method. In this approach, the execution of the methods is done at the level of the Hibernate layer. Therefore, the computation of the fulfilment degree is done outside the database system.

As a consequence of this work, another implementation has been proposed on the top of the Oracle database system (See publication list, reference 19). This implementation is a module that extend the functionality of the fuzzy object-relational server which implements the GEFRED database model. Aspects like the modelling of imperfect time intervals and the redefinition of the DML language are covered in this implementation. The theoretical model is based on the model presented in Chapter 3, but with some modifications to emulate the behaviour of the Workspace Manager by Oracle.

7.3. General Conclusion and Further Research Work

In this research work, we have investigated the modelling of imperfect temporal information in an Information System. First we developed a complete formal framework to represent and handle imperfect temporal information. This framework has been proven to be sound and consistent with possibility theory. According with the result of the thesis, the framework models imperfect time intervals as well as the relationships between them. Compared to other approaches this framework provides a constraint-based system in which the full relational information is kept. The framework, here applied to time intervals, has been extended to deal also with discrete possibility distributions.

Based on the theoretical framework to model and handle temporal information, we developed a theoretical model for valid-time relational databases. The model is build on the top of GEFRED and implements the representation of time-dependent objects in a relational database. Compared to other theoretical temporal database models, in this proposal, the validity period may contain imperfection / imprecision / uncertainty. Despite the storage of imperfect information, the model offers a consistence mechanism that ensures the consistency of the time-dependent objects / tuples stored in the database.

The querying of a temporal database has been usually done by extending the relational algebra with the Allen's operators. In this thesis we go beyond, by extending the satisfaction-dissatisfaction approach. A temporal constraint consists on an Allen relation and a time interval. The time interval might contain imprecision and can be specified by means of a Possibilistic Time period (PVP). We consider two ways of querying by using a temporal constraint. The first method adds a global temporal constraint which sets a temporal frame that the selected tuples have to fulfil. In the second

method, we propose to set a temporal constraint inside each elementary criteria. Hence, each attribute may contains a temporal frame that the selected tuples have to fulfil.

Both method present the issue of the aggregation between the temporal and the non-temporal constraints. In this thesis we have proposed a method for both aggregation and ranking. By default, both constraints have the same weight when aggregating although the weight can be adjusted.

Further research work in bipolar querying of temporal databases is currently active. The specification of a bipolar temporal constraint will add even more expressive power to the queries. This is of special interest for certain applications such as criminal investigation. Again, one of the main problems is the aggregation and the ranking of the results obtained in the query.

In this thesis is also presented a novelty visualization method for time intervals. This method is based on the triangular model TM which represents in a two-dimensional model a time interval, which is usually represented in a one-dimensional line. The visualization method extends the TM to visualize uncertain time intervals UTIs. By using this visualization, the Allen's relations among uncertain time intervals can be obtained at a glance. The main benefit of this representation is the possibility to visualize big amounts of time intervals and the relationships between them.

Finally, we provide an implementation in an open source framework for the theoretical model for temporal databases presented in this thesis. The main benefits of this implementation is that it is portable. The representation, querying and the consistence mechanism presented in the theoretical model have been developed in this implementation. Another implementation on the top of the Oracle database have been done by using an extension of the theoretical model of this thesis. Further research work will include the implementation of bipolar querying and the temporal constraints. Also the visualization of temporal results by using the triangular model.

A

Possibilistic evaluation of sets

The contents of this appendix have been partially published on:

- A. Bronselaer, J. E. Pons, G. De Tré, and O. Pons, "Possibilistic evaluation of sets," *Int. J. Uncertainty Fuzziness Knowlege-Based Syst.*, vol. 21, no. 3, pp.325–346, 2013.

Contents

In the past decades, the theory of possibility has been developed as a theory of uncertainty that is compatible with the theory of probability. Whereas probability theory tries to quantify uncertainty that is caused by variability (or equivalently randomness), possibility theory tries to quantify uncertainty that is caused by incomplete information. A specific case of incomplete information is that of ill-known sets, which is of particular interest in the study of temporal databases. However, the construction of possibility distributions in the case of ill-known sets is known to be overly complex.

In this appendix, we present a framework for dealing with the evaluation of constraints defined by ill-known values.

The appendix is organized as follows. Section A.1 shows how regular sets can be evaluated by using ill-known constraints. In Section A.2, the proposed mechanism is applied to the evaluation of intervals. The proposal is compared to the approach by Dubois and Prade [96]. It is shown that the results obtained are consistent with the ones of Dubois and Prade. There are other approaches in the literature for dealing with time intervals, but it is shown that these approaches imply a lost of knowledge.

A.1. Set evaluation by ill-known constraints

In this section, we shall introduce the theoretical framework in which we reason throughout the rest of this paper. The motivation for our framework is found in the early work of Dubois and Prade on ill-known sets [154]. In that work, Dubois and Prade provide a treatment of what they call *incomplete conjunctive information* and handle the problem of *ill-known sets*, i.e. crisp sets that are partially unknown due to incomplete information. They argue in their work that a possibility distribution over a universe $\mathcal{P}(U)$ can be considered as a heavy-to-handle representation, due to the exponential complexity in terms of the size of the universe. It can be reasoned that this high complexity stems from the desire to represent possibilistic information about *all* sets in $\mathcal{P}(U)$. At this point, the framework that is presented within this paper, differs from the framework by Dubois and Prade. Rather than providing all possibilistic information, it is assumed that there are a (limited) number of alternatives about which possibilistic information is required. More specific, the question that is aimed to answer here is the following: "Given a set A, what is the certainty that A meets some requirements?". When providing an answer to this question, it will be explicitly taken into account that:

- a requirement on a set can be expressed in terms of requirements on *elements*,
- *uncertainty* about such requirements can exist.

Note that the specification of sets by requirements on elements, is a very natural concept in mathematics if we think of the set builder notation $\{u \in U|.\}$. The aim of this paper is to take this familiar concept and to extend it towards a more general case that allows uncertainty in the specification of requirements. In the remainder of the paper, such uncertain requirements will be referred to as *ill-known constraints*.

Before giving any technical details, let us first provide some practical examples that require the checking of ill-known constraints. As a first example, consider the set of real numbers $\mathbb{R}$ and consider a crisp interval $[a, b]$. A fundamental problem in fuzzy temporal databases is to find out how this interval is positioned with respect to the interval $[X, Y]$. Hereby, X represents the ill-known start point of the interval and Y represents the ill-known end point of the interval[1]. Let us make this example more

[1]Note that $[X, Y]$ is thus *not* a fuzzy interval (Section 3.1.3). As such, the problem described here is semantically different from fuzzifications of the Allen relations [9].

specific and suppose that it is required to know whether or not $[a, b]$ is a subset of $[X, Y]$. In order to answer this question, two constraints must be verified. The first constraint states that *all* elements in $[a, b]$ must be larger than or equal to X, while the second constraint states that *all* elements in $[a, b]$ must be smaller than or equal to Y. Moreover, *both* constraints must be satisfied in order for $[a, b]$ to be a subset of $[X, Y]$. Consequently, this problem has a conjunctive nature with respect to satisfaction of the constraints. From this example, two observations can be made. Firstly, it can be seen that each constraint on the set $[a, b]$ is specified as a constraint on the elements of $[a, b]$. More specific, this specificiation is obtained by means of a binary relation R and an ill-known value X. For example, the first constraint requires that all elements from $[a, b]$ are in relation $\geq$ to the ill-known value X. Secondly, a positive evaluation of a set requires the satisfaction of several constraints. This means that Boolean reasoning is required to allow the verification of aggregate (i.e. complex) constraints.

While, this first example represents a class of problems where the universe of discourse is equipped with a total order, the aim of this paper is to provide a more general framework that can be applied to *any* universe of discourse. Therefore, as a second example, consider the set of languages $\mathbb{L}$ and assume that m language experts $E_1, ..., E_m$ are available. Suppose that each of these experts are provided with the question: "What is the main language that John speaks?". Because the answer to this question can be uncertain, each expert E_i provides a possibilistic variable X_i as an answer to the question. Suppose that we are encountered with the question: "What is the possibility that John speaks English, French and Spanish?". To answer this question, a decision rule is required such as: "John speaks languages $\{l_1, ..., l_k\}$ if each language is confirmed by at least one expert". Unfortunately, a mechanism to evaluate this decision rule from the given variables $X_1, ..., X_m$ is not at hand within standard possibility theory. It is however possible to translate the rule into a set of constraints, stating that John speaks languages $\{l_1, ..., l_k\}$ if, for any expert E_i, there exists a language l_j such that $l_j = X_i$ or, equivalently, not all elements in $\{l_1, ..., l_k\}$ are different from X_i. Again, it is observed that a constraint is specified on elements by means of an ill-known value (i.e. X_i) and a binary relation (i.e. $=$). Again, it is observed that the evaluation of a set stems from a Boolean combination of several constraints.

With these examples in mind, let us begin with the definitions of some basic concepts. As observed in the above mentioned examples, a constraint is specified by means of a binary relation R and a value x. For clarity, we first provide a definition in the crisp case, i.e. the case where constraints are *not* ill-known.

Definition 105. Given a universe U, a constraint C on a set $A \subseteq U$ is specified by means of a binary relation $R \subseteq U^2$ and a fixed value $x \in U$, i.e.:

$$C \triangleq (R, x)\,. \tag{A.1}$$

It is said that a set A satisfies the constraint C if and only if:

$$\forall a \in A : (a, x) \in R. \tag{A.2}$$

Definition 105 adheres to the fact that the framework of constraint evaluation is Boolean in nature. In order to make an explicit connection with the Boolean framework, we shall adopt the notation $C(A)$ to indicate a Boolean proposition which is true if A satisfies C and which is false if A fails (i.e. does not satisfy) C.

Example 42. Consider the set of natural numbers $\mathbb{N}$ and consider the constraint $C = (\leq, 3)$, then the set $A = \{1, 2, 3\}$ satisfies constraint C because all elements in A are in relation $\leq$ to the value 3.

Example 42 illustrates that crisp constraints are quite trivial. However, the triviality disappears when the step towards ill-known constraints is made. An ill-known constraint differs from a constraint in the sense that the value in the constraint specification is no longer a crisp value x, but an ill-known value X. This generalization requires a mechanism for checking whether or not a crisp value is in relation to an ill-known value. Such a mechanism is provided by application of Zadeh's Extension Principle. More specific, for any ill-known value X over U and for any binary relation R over U, we have that:

$$\forall u \in U : \mathrm{Pos}((u, X) \in R) = \sup_{(u,w)\in R} \pi_X(w) \tag{A.3}$$

$$\forall u \in U : \mathrm{Nec}((u, X) \in R) = \inf_{(u,w)\notin R} 1 - \pi_X(w). \tag{A.4}$$

With this mechanism at hand, it is possible to define the concept of an ill-known constraint.

Definition 106. Given a universe U, an ill-known constraint C on a set $A \subseteq U$ is specified by means of a binary relation $R \subseteq U^2$ and an ill-known value X, i.e.:

$$C \triangleq (R, X)\,. \tag{A.5}$$

The uncertainty that a set $A \subseteq U$ satisfies C is given by:

$$\mathrm{Pos}(C(A)) = \min_{a\in A}\Big(\mathrm{Pos}(a, X) \in R\Big) = \min_{a\in A}\Big(\sup_{(a,w)\in R} \pi_X(w)\Big) \tag{A.6}$$

$$\mathrm{Nec}(C(A)) = \min_{a\in A}\Big(\mathrm{Nec}(a, X) \in R\Big) = \min_{a\in A}\Big(\inf_{(a,w)\notin R} 1 - \pi_X(w)\Big). \tag{A.7}$$

Note that the possibility (resp. necessity) that A satisfies C is given by the possibility (resp. necessity) that *all* elements in A are in relation R to X. The quantifier "all" is hereby modeled by the minimum operator, as is prescribed by the rules of possibility theory.

The values $\mathrm{Pos}(C(A))$ and $1 - \mathrm{Nec}(C(A))$ together constitute a possibility distribution $\pi_{C(A)}$ over the set of Boolean values $\mathbb{B}$ (in literature also known as a possibilistic truth value [37, 255, 256]). This is formalized in the following theorem.

Theorem 2. Given a universe U and an ill-known constraint C specified by the ill-known value X and a binary relation R, then for any $A \subseteq U$, $\pi_{C(A)}$ is a possibility distribution over the set of Boolean values $\mathbb{B}$.

Proof. Let us assume a set $A \subseteq U$. It is sufficient to prove that $\pi_{C(A)}$ is normalized, which means that we must prove that either $\pi_{C(A)}(T)$ or $\pi_{C(A)}(F)$ equals 1. On the one hand we have that:

$$\pi_{C(A)}(T) = \min_{a\in A}\left(\sup_{(a,w)\in R} \pi_X(w)\right). \tag{A.8}$$

On the other hand we have that:

$$\pi_{C(A)}(F) = 1 - \min_{a\in A}\left(\inf_{(a,w)\notin R} 1 - \pi_X(w)\right) \tag{A.9}$$

which can be rewritten as:

$$\pi_{C(A)}(F) = \max_{a \in A} \left(\sup_{(a,w) \notin R} \pi_X(w) \right). \tag{A.10}$$

In case (1), let us assume that:

$$\forall a \in A : \exists (a, v) \in R : \pi_X(v) = 1. \tag{A.11}$$

If this assumption holds, then it can be easily seen that:

$$\pi_{C(A)}(T) = 1. \tag{A.12}$$

In case (2), the above-made assumption does not hold, which means that:

$$\exists a' \in A : \neg \left(\exists (a', v) \in R : \pi_X(v) = 1 \right). \tag{A.13}$$

However, due to the fact that π_X is a possibility distribution (and thus normalized), we have that:

$$\exists v' \in U : \pi_X(v') = 1. \tag{A.14}$$

This means that there exists an $a' \in A$ and a $v' \in U$, such that $(a', v') \notin R$ and $\pi_X(v') = 1$. Consequently:

$$\pi_{C(A)}(F) = 1. \tag{A.15}$$

Considering the fact that either the above-made assumption holds or not, $\pi_{C(A)}$ is proven to be normalized. □

Example 43. Consider the set of natural numbers $\mathbb{N}$ and consider the constraint $C = (\leq, X)$ where X is an ill-known value specified by the possibility distribution π_X as shown in Figure A.1.

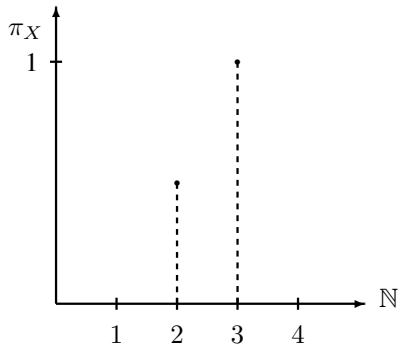


Figure A.1: Possibility distribution of X

The uncertainty about whether or not set $A = \{1, 2, 3\}$ satisfies C is given by:

$$\text{Pos}(C(A)) = \min_{a \in \{1,2,3\}} \left(\sup_{a \leq w} \pi_X(w) \right) = 1 \tag{A.16}$$

$$\text{Nec}(C(A)) = \min_{a \in \{1,2,3\}} \left(\inf_{a > w} 1 - \pi_X(w) \right) = 0.5. \tag{A.17}$$

So far, we have shown how it can be verified whether or not a set satisfies an ill-known constraint. However, from the examples at the beginning of this section, it is observed that Boolean combinations of constraints are required. For example, the problem of interval evaluation as explained earlier requires that all elements of an interval $[a, b]$ are larger than a value X and at the same time smaller than a value Y, which implies that a conjunctive Boolean combination of both constraints must be satisfied. To allow Boolean combinations of constraints, the following definitions are introduced.

Definition 107. Consider a universe U, an n-ary vector $\mathbf{C}$ of constraints and a Boolean function $\mathcal{B} : \mathbb{B}^n \to \mathbb{B}$. An evaluation function is defined by:

$$\lambda : \mathcal{P}(U) \to \mathbb{B} : A \mapsto \mathcal{B}\Big(C_1(A), ..., C_n(A)\Big). \tag{A.18}$$

Definition 107 presents the definition of an evaluation function that evaluates a Boolean combination of some basic constraints. Informally, it states that a set A passes the evaluation made by λ if the Boolean combination of some propositions equals T. This crisp definition can be generalized to the case of ill-known constraints.

Definition 108. Consider a universe U, an n-ary vector $\mathbf{C}$ of ill-known constraints and a Boolean function $\mathcal{B} : \mathbb{B}^n \to \mathbb{B}$. The uncertainty about the evaluation of a set A by an evaluation function λ is then given by:

$$\forall A \in \mathcal{P}(U) : \pi_{\lambda(A)} = \widetilde{\mathcal{B}}\Big(\pi_{C_1(A)}, ..., \pi_{C_n(A)}\Big) \tag{A.19}$$

Hereby, $\widetilde{\mathcal{B}}$ is the possibilistic extension of $\mathcal{B}$.

It is well known that any Boolean function $\mathcal{B}$ can be cast to a canonical form [178], requiring only the logical conjunction $\wedge$, logical disjunction $\vee$ and logical negation. Therefore, only the case of Boolean conjunction, Boolean disjunction and Boolean negation will be treated within the scope of this paper. By applying the possibilistic extensions of $\wedge$, $\vee$ and $\neg$, concrete equations are obtained for the calculations of uncertainty about the evaluation of a set by means of an evaluation function λ. In the case of conjunction (i.e., $\mathcal{B} = \wedge$), the inference of uncertainty about the evaluation of a set reduces to:

$$\forall A \in \mathcal{P}(U) : \mathrm{Pos}(\lambda(A)) = \min_{i=1}^{n} \mathrm{Pos}\left(C_i(A)\right) \tag{A.20}$$

$$\forall A \in \mathcal{P}(U) : \mathrm{Nec}(\lambda(A)) = \min_{i=1}^{n} \mathrm{Nec}\left(C_i(A)\right). \tag{A.21}$$

In the case of disjunction (i.e. $\mathcal{B} = \vee$), the inference of uncertainty about the evaluation of a set reduces to:

$$\forall A \in \mathcal{P}(U) : \mathrm{Pos}(\lambda(A)) = \max_{i=1}^{n} \mathrm{Pos}\left(C_i(A)\right) \tag{A.22}$$

$$\forall A \in \mathcal{P}(U) : \mathrm{Nec}(\lambda(A)) = \max_{i=1}^{n} \mathrm{Nec}\left(C_i(A)\right). \tag{A.23}$$

Note that by using the functions $\min$ and $\max$ here, there is an implicit assumption that the possibilistic variables π_{C_i} are mutual $\min$-dependent in the sense of De Cooman (i.e. non-interactive). For an extensive reading on (in)dependency of possibilistic variables, the reader is referred to [172–174]. In case of $\neg$, we get:

$$\forall A \in \mathcal{P}(U) : \mathrm{Pos}(\neg\lambda(A)) = 1 - \mathrm{Nec}(\lambda(A)) \tag{A.24}$$

$$\forall A \in \mathcal{P}(U) : \mathrm{Nec}(\neg\lambda(A)) = 1 - \mathrm{Pos}(\lambda(A)). \tag{A.25}$$

Let us provide an example that illustrates the use of Definition 108.

Example 44. Consider a universe $U = \{a, b, c, d\}$ and consider two binary relations over U as shown in Figure A.2.

R_1	a	b	c	d
a	x			
b		x		
c			x	
d				x

R_2	a	b	c	d
a	x	x		
b	x	x		
c			x	x
d			x	x

Figure A.2: Two binary relations on U

Note that R_1 is the equality relation over U and R_2 is an equivalence relation over U. Let us also consider two ill-known values X_1 and X_2 for which the possibility distributions are shown in Figure A.3.

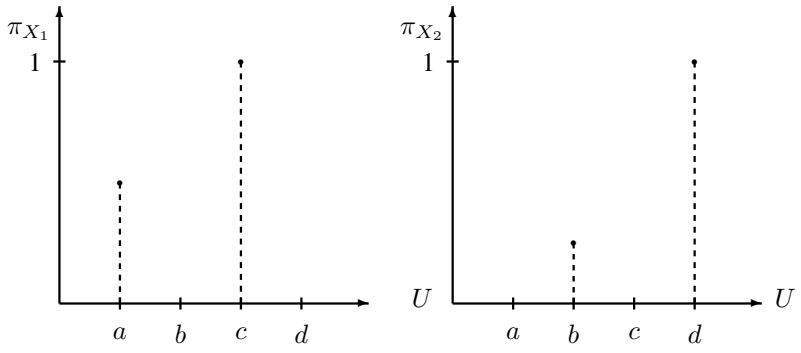


Figure A.3: Possibility distributions of X_1 and X_2

Based on these distributions, it is possible to infer the uncertainty about the ill-known constraints C_1 and C_2. For the first constraint, we find that:

$$\text{Pos}((a, X_1) \in R_1) = 0.5 \qquad \text{Nec}((a, X_1) \in R_1) = 0 \tag{A.26}$$
$$\text{Pos}((b, X_1) \in R_1) = 0 \qquad \text{Nec}((b, X_1) \in R_1) = 0 \tag{A.27}$$
$$\text{Pos}((c, X_1) \in R_1) = 1 \qquad \text{Nec}((c, X_1) \in R_1) = 0.5 \tag{A.28}$$
$$\text{Pos}((d, X_1) \in R_1) = 0 \qquad \text{Nec}((d, X_1) \in R_1) = 0. \tag{A.29}$$

For the second constraint, we find that:

$$\text{Pos}((a, X_2) \in R_2) = 0.25 \qquad \text{Nec}((a, X_2) \in R_2) = 0 \tag{A.30}$$
$$\text{Pos}((b, X_2) \in R_2) = 0.25 \qquad \text{Nec}((b, X_2) \in R_2) = 0 \tag{A.31}$$
$$\text{Pos}((c, X_2) \in R_2) = 1 \qquad \text{Nec}((c, X_2) \in R_2) = 0.75 \tag{A.32}$$
$$\text{Pos}((d, X_2) \in R_2) = 1 \qquad \text{Nec}((d, X_2) \in R_2) = 0.75. \tag{A.33}$$

Now assume an evaluation function λ that evaluates the Boolean disjunction ($\mathcal{B} = \vee$) of C_1 and C_2. For any set $A \subseteq U$, the possibility and necessity that A passes the evaluation through λ is shown in Table A.1.

A	$\mathrm{Pos}(\lambda(A))$	$\mathrm{Nec}(\lambda(A))$
$\emptyset$	1	1
$\{a\}$	0.5	0
$\{b\}$	0.25	0
$\{c\}$	1	0.75
$\{d\}$	1	0.75
$\{a, b\}$	0.25	0
$\{a, c\}$	0.5	0
$\{a, d\}$	0.5	0
$\{b, c\}$	0.25	0
$\{b, d\}$	0.25	0
$\{c, d\}$	1	0.75
$\{a, b, c\}$	0.25	0
$\{a, b, d\}$	0.25	0
$\{a, c, d\}$	0.5	0
$\{b, c, d\}$	0.25	0
$\{a, b, c, d\}$	0.25	0

Table A.1: Uncertainty about set evaluation

A.2. An application on intervals

In this section, the proposed reasoning is more deeply applied to the specific context of intervals on the real line. It will be shown in this section that the general framework that is presented here, is consistent with results of earlier work concerning interval reasoning. The setting of intervals is of specific interest in the context of fuzzy temporal databases. A temporal database [73] is a database that manages some aspects of time in its schema. The time can be represented either as points or intervals [84]. Fuzzy temporal models [23] have been proposed when the time points [96] or intervals [24] are ill-known. Allen [9] defined thirteen possible relations between two crisp time intervals. For fuzzy intervals, several proposals [19, 22, 23] have been done.

As explained above, in temporal databases, points in time can be ill-known, i.e. a point in time is modelled as an ill-known value. When two such ill-known points are given, an interesting problem is the inference of uncertainty about the interval that is enclosed by these two ill-known points. In the proposed framework, this problem can be solved by evaluating an interval against two ill-known constraints, where the binary relations are ordering relations on the set of real numbers. We first treat this special case of interval evaluation separately, because some authors seem to solve this problem by transforming the given ill-known points into a fuzzy set. However, it is shown here why such a solution fails. In a second step, it will be shown show how a crisp interval can be compared to an ill-known interval by using Allen relations.

A.2.1. Interval evaluation by ill-known constraints

Consider two ill-known values X and Y on the set of real numbers $\mathbb{R}$. Uncertainty about the values taken by X and Y is given by possibility distributions π_X and π_Y. The problem that is studied first here, is how uncertainty about the fact that a crisp interval is enclosed by the interval with boundary points X and Y can be inferred. More concretely, for a crisp interval $I = [a, b]$, we want to know whether all points

in this interval reside between the boundaries X and Y. This means that, assuming X specifies the lower bound and Y the upper bound, we want to known whether all points in the interval are larger than or equal to X and smaller than or equal to Y. This can be done by casting the general framework introduced in the previous section into this problem. Therefore, we consider two ill-known constraints.

$$C_1 \triangleq (\geq, X) \tag{A.34}$$

$$C_2 \triangleq (\leq, Y)\,. \tag{A.35}$$

Applying the inference of uncertainty as proposed in our general reasoning, we find for the first constraint that:

$$\mathrm{Pos}\left(C_1([a,b])\right) = \min_{r\in[a,b]}\left(\sup_{r\leq w}\pi_X(w)\right) \tag{A.36}$$

$$\mathrm{Nec}\left(C_1([a,b])\right) = \min_{r\in[a,b]}\left(\inf_{r>w} 1-\pi_X(w)\right) \tag{A.37}$$

which can be simplified to:

$$\mathrm{Pos}\left(C_1([a,b])\right) = \sup_{a\leq w}\pi_X(w) \tag{A.38}$$

$$\mathrm{Nec}\left(C_1([a,b])\right) = \inf_{a>w} 1-\pi_X(w). \tag{A.39}$$

For the second constraint, we find that:

$$\mathrm{Pos}\left(C_2([a,b])\right) = \min_{r\in[a,b]}\left(\sup_{r\geq w}\pi_Y(w)\right) \tag{A.40}$$

$$\mathrm{Nec}\left(C_2([a,b])\right) = \min_{r\in[a,b]}\left(\inf_{r<w} 1-\pi_Y(w)\right) \tag{A.41}$$

which can be simplified to:

$$\mathrm{Pos}\left(C_2([a,b])\right) = \sup_{b\geq w}\pi_Y(w) \tag{A.42}$$

$$\mathrm{Nec}\left(C_2([a,b])\right) = \inf_{b<w} 1-\pi_Y(w). \tag{A.43}$$

The uncertainty about the inclusion of an interval $I = [a,b]$ in the interval with ill-known boundaries can now be found by evaluating $[a,b]$ against the evaluation function λ with $\mathcal{B} = \wedge$. Application of (3.25) and (3.26) leads to:

$$\mathrm{Pos}\left(\lambda([a,b])\right) = \min\left(\mathrm{Pos}(C_1([a,b])), \mathrm{Pos}(C_2([a,b]))\right) \tag{A.44}$$

$$\mathrm{Nec}\left(\lambda([a,b])\right) = \min\left(\mathrm{Nec}(C_1([a,b])), \mathrm{Nec}(C_2([a,b]))\right). \tag{A.45}$$

These last expressions can also be expanded as:

$$\mathrm{Pos}\left(\lambda([a,b])\right) = \min\left(\sup_{a\leq w}\pi_X(w), \sup_{b\geq w}\pi_Y(w)\right) \tag{A.46}$$

$$\mathrm{Nec}\left(\lambda([a,b])\right) = \min\left(\inf_{a>w} 1-\pi_X(w), \inf_{b<w} 1-\pi_Y(w)\right). \tag{A.47}$$

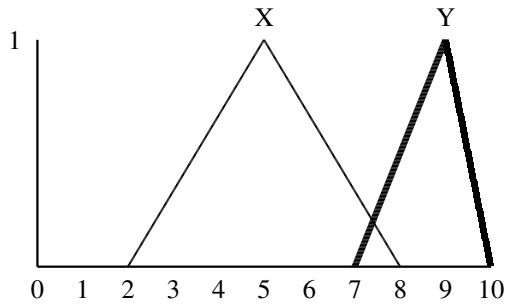


Figure A.4: The fuzzy numbers X and Y.

Note that the interval $[X, Y]$ used here, is certainly not a fuzzy interval. Instead, we are dealing with an ill-known interval, i.e. it is a crisp interval, but it is partially unknown which values are in this interval. The uncertainty stems from the fact that the interval boundaries are ill-known. These ideas get more clear in the following example.

Example 45. To illustrate the above mentioned mechanism, consider the fuzzy numbers X and Y for which the membership function can be regarded as a possibility distribution. The membership functions are given by:

$$\begin{aligned} X &= [5, 3, 3] \\ Y &= [9, 2, 1] \end{aligned} \tag{A.48}$$

and are shown in Figure A.4. The knowledge about the evaluation of an interval $[a, b]$ is modelled by the expressions in (A.46) and (A.47). Figure A.5 shows a 3D plot of the possibility that an interval $[a, b]$ passes the evaluation (i.e. given by (A.46)). Note the triangular form for the resulting possibility distribution since the condition $a \leq b$ holds. The necessity plot (i.e. given by (A.47)) is obtained in a similar way and is shown in

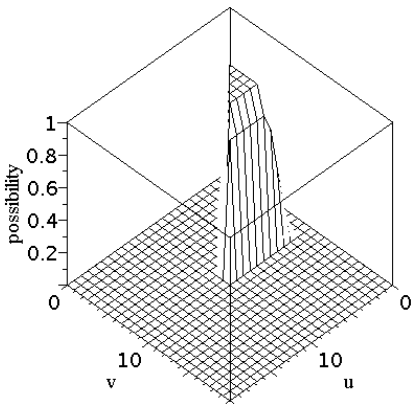


Figure A.5: Possibility of evaluation for the interval $[a, b]$.

Figure A.6. Notice that the necessity plot is not normalized because the supports of X and Y overlap.

It can be seen that the evaluation of intervals obtained by the presented constraints in fact evaluates whether a crisp interval is inside two ill-known boundaries. Hence, an

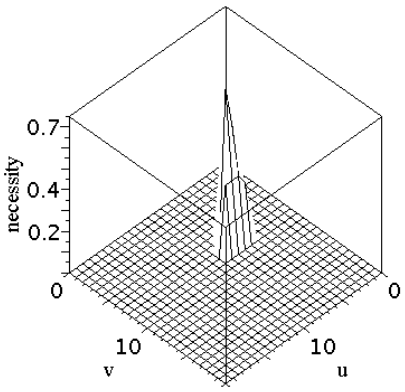


Figure A.6: Necessity of evaluation for the interval $[a, b]$.

evaluation of the relation "during or equal" in the sense of Allen is obtained [9]. It is noted that any of the Allen relations (or a combination of them) can be evaluated in a similar way. This statement is proven here by providing the basic constraints and the corresponding function $\mathcal{B}$, for the seven basic Allen relations (the others are merely inversions of the discussed relations). Assume a crisp interval $I = [a, b]$ on the one hand and an interval J with ill-known boundaries X and Y. The uncertainty about X and Y is given by the possibility distributions π_X and π_Y. For any of the Allen relations, we can compare I with J by formulating a set of constraints and a Boolean function. Table A.2 shows the seven basic interval relations proposed by Allen with the corresponding constraints in the framework and the corresponding Boolean function. When taking a closer look at Table A.2, it can be noticed that in some cases it is not required that all elements of a set A satisfy a constraint. For example, the Allen relation I equals J requires to evaluate that both the lower and upper boundaries of both intervals are equal. However, the definition of a constraint (Definition 105) implies that a constraint is satisfied if *all* elements of a set are in relation to the given threshold x. It can thus be questioned whether the framework should also allow to evaluate whether *at least one element* is in relation to a given threshold x (i.e. a notion of the $\exists$ quantifier). The study of the Allen relations in Table A.2 shows however that such a notion is already at hand in the presented framework. This can be shown on a more formal level. Suppose that for a set A, an evaluation of the following kind is required:

$$\exists a \in A : (a, x) \in R \tag{A.49}$$

then we can rewrite this as

$$\neg\Big(\forall a \in A : \neg\big((a, x) \in R\big)\Big) \tag{A.50}$$

which is equivalent to

$$\neg\Big(\forall a \in A : (a, x) \notin R\Big) \tag{A.51}$$

and finally we have that

$$\neg\Big(\forall a \in A : (a, x) \in \overline{R}\Big). \tag{A.52}$$

This provides us with a similar construction as in Definition 105, with the only difference that the constraint is negated. However, the possibilistic extension of the operator

Allen Relation	Constraints	$\mathcal{B}\left(C_1(I), ..., C_n(I)\right)$
I before J	$C_1 \triangleq (<, X)$	$C_1(I)$
I equal J	$C_1 \triangleq (\geq, X)$ $C_2 \triangleq (\neq, X)$ $C_3 \triangleq (\leq, Y)$ $C_4 \triangleq (\neq, Y)$	$C_1(I) \wedge \neg C_2(I) \wedge C_3(I) \wedge \neg C_4(I)$
I meets J	$C_1 \triangleq (\leq, X)$ $C_2 \triangleq (\neq, X)$	$C_1(I) \wedge \neg C_2(I)$
I overlaps J	$C_1 \triangleq (<, Y)$ $C_2 \triangleq (\leq, X)$ $C_3 \triangleq (\geq, X)$	$C_1(I) \wedge \neg C_2(I) \wedge \neg C_3(I)$
I during J	$C_1 \triangleq (>, X)$ $C_2 \triangleq (\leq, Y)$ $C_3 \triangleq (\geq, X)$ $C_4 \triangleq (<, Y)$	$\big(C_1(I) \wedge C_2(I)\big) \vee \big(C_3(I) \wedge C_4(I)\big)$
I starts J	$C_1 \triangleq (\geq, X)$ $C_2 \triangleq (\neq, X)$	$C_1(I) \wedge \neg C_2(I)$
I finishes J	$C_1 \triangleq (\leq, Y)$ $C_2 \triangleq (\neq, Y)$	$C_1(I) \wedge \neg C_2(I)$

Table A.2: Allen's relations represented in the framework.

$\neg$ can be used for evaluation of such constructions. It might be noticed that evaluations where only one element must satisfy a constraint could be defined directly in the framework, perhaps leading to more simple formulas in Table A.2. The authors have chosen not to do so, in order to keep the basic framework as simple as possible.

In what follows, the connection with the work of some other authors is presented, hereby adopting the notations from the works that are referred.

A.2.2. Dubois and Prade's approach

In this subsection, a comparison with the work of Dubois and Prade is made from two points of view.

Firstly, it can be seen that the application to (temporal) intervals of our framework provides equivalent results as the ones by Dubois and Prade in their work on temporal reasoning [96]. This demonstrates to a certain level the correctness of the framework presented in this paper. More specific, starting from a general framework of reasoning about sets, we obtain equivalent results than those obtained by a specific formulation of the problem. As a result, the reasoning about uncertainty is consistent with an alternative reasoning. However, note that in the presented framework, we have an explicit connection with Boolean logic as the reasoning takes place in the framework of possibilistic truth values. This reasoning is not presented by Dubois and Prade.

Secondly, our framework avoids the usage of a possibility distribution over the set

$\mathcal{P}(U)$ as is suggested in [154]. This provides a more simple reasoning. In addition, aspects such as monotonicity are implicitly taking into account. For example, let us consider two crisp intervals $[a, b]$ and $[c, d]$ with the constraint that $[a, b] \subset [c, d]$. Clearly, with $[X, Y]$ an ill-known interval, the possibility that $[a, b]$ is a subset of $[X, Y]$ must be greater than or equal to the possibility that $[c, d]$ is a subset of $[X, Y]$. If a possibility distribution is to be constructed over $\mathcal{P}(\mathbb{R})$, such monotonicity has to be taken into account explicitly, while in the presented framework, such monotonicity is implicitly taken into account.

In conclusion, within this paper, a general framework for reasoning about the uncertainty of sets is presented that is more suitable than the framework of ill-known sets as presented by Dubois and Prade [154]. However, when applying the framework to the specific case of intervals over $\mathbb{R}$, the obtained results are equivalent and consistent with earlier results [154].

A.2.3. Comparison with fuzzy transformations

Next to Dubois and Prade, there are several alternative proposals to deal with uncertainty about intervals. These proposals initiate from the idea that two fuzzy numbers that represent an ill-known interval, can be transformed into a fuzzy interval. Two such transformations are based on the convex hull and on the transformation preserving the imprecision [24]. Both transformations are illustrated in Figure A.7.

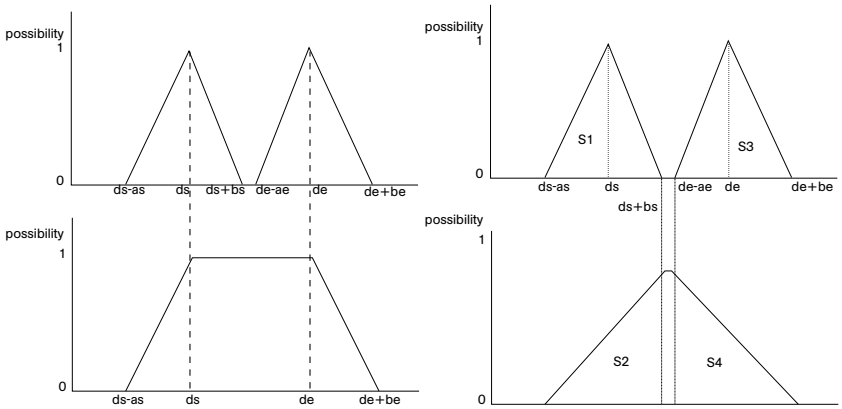


Figure A.7: Transformations of ill-known values.

Approaches that adopt such transformations are however not consistent with possibility theory. We provide two arguments to illustrate this:

1. The fuzzy sets that model the two fuzzy numbers are possibility distributions over $\mathbb{R}$ because they both describe an ill-known point on the real line. However, the fuzzy set obtained after the transformation is still a possibility distribution over $\mathbb{R}$ and not over $\mathcal{P}(\mathbb{R})$. This means that the approaches involving a transformation fail to establish possibilistic information on the level of *intervals*.

2. If the obtained fuzzy after the transformation is to be treated as a fuzzy *interval*, then the notion of possibility is suddenly dropped. Such a casting from possibility theory to fuzzy set theory lacks a theoretical foundation in the literature of fuzzy set theory. Moreover, it involves a loss of information. At best, like with

the convex hull transformation, the membership function provided for the fuzzy interval preserves the possibility. However, information about necessity is never preserved.

A.3. Conclusions

Possibility theory is a theory of uncertainty suited for the treatment of uncertainty caused by incomplete information. It has been studied and developed over the past decades. Unfortunately, there seems to be many misinterpretations of this theory when it is applied to the case of sets. Dubois and Prade have shown that the correct treatment of sets would imply a highly complex possibility distribution. Elaborating on their work, we have proposed an elegant and novel method for the inference and modelling of uncertainty about crisp sets. A framework for set evaluation is proposed where a set is evaluated againts a collection of constraints, which can be ill-known. As such, we develop a framework in which uncertainty on the level of elements is propagated to uncertainty on the level of sets. The uncertainty is expressed by means of possibility distributions over the Boolean domain $\mathbb{B}$. An interesting application of the proposed framework, is the case of interval evaluation, which is for example useful in temporal databases. We have shown that (i) our framework allows to evaluate the well known Allen relations, (ii) our framework is compatible with the results by Dubois and Prade in the field of uncertain intervals and (iii) some techniques from literature with respect to fuzzy intervals are not fully correct in the sense of possibility theory.

Bibliography

[1] "Gregorian calendar." http://en.wikipedia.org/wiki/Gregorian_calendar.

[2] G. Shackle, *Decision, order and time in human affairs*. Cambridge University Press, 1961.

[3] W. Klein, *Time in Language*. London, U.K.: Routledge, 1994.

[4] F. Devos, N. Van Gyseghem, R. Vandenberghe, and R. De Caluwe, "Modelling vague lexical time expressions by means of fuzzy set theory," *Journal of Quantitative Linguistics*, vol. 1, no. 3, pp. 189–194, 1994.

[5] F. Devos, P. Maesfranckx, and G. De Tré, "Granularity in the interpretation of around in approximative lexical time indications," *Journal of Quantitative Linguistics*, vol. 5, pp. 167–173, 1998.

[6] R. De Caluwe, B. Van der Cruyssen, G. De Tré, F. Devos, and P. Maesfranckx, *Fuzzy time indications in natural languages interfaces*, pp. 163–185. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[7] A. Bolour, T. L. Anderson, L. J. Dekeyser, and H. K. T. Wong, "The role of time in information processing: a survey," *ACM SIGMOD Record*, vol. 12, pp. 27–50, April 1982.

[8] B. Van der Cruyssen and G. De Caluwe, R.and De Tré, "A theoretical fuzzy time model based on granularities," *EUFIT'97*, pp. 1127–1131, Sep 1997.

[9] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, pp. 832–843, 1983.

[10] L. Fiordi, "Semantic conceptions of information.." The Standford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/information-semantic/, 2005.

[11] E. Wikipedia, "Information." http://en.wikipedia.org/wiki/Information. Accessed on June 2012.

[12] "Definition of information system." http://en.wikipedia.org/wiki/Information_systems, 09 2007. Accessed on May 2013.

[13] "Information system discipline." http://en.wikipedia.org/wiki/Information_systems_%28discipline%29. Accessed on June 2012.

[14] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, pp. 377–387, June 1970.

[15] E. F. Codd, "Extending the database relational model to capture more meaning," *ACM Transactions on Database Systems*, vol. 4, pp. 397–434, 1979.

[16] J. Melton and A. R. Simon, *Understanding the new SQL: a complete guide*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[17] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[18] D. Mitra and et al., "A possibilistic interval constraint problem: Fuzzy temporal reasoning," in *Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proc. of the Third IEEE Conference on*, vol. 2, pp. 1434–1439, jun 1994.

[19] G. Nagypál and B. Motik, "A fuzzy model for representing uncertain, subjective, and vague temporal knowledge in ontologies," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, vol. 2888 of *LNCS*, pp. 906–923, Springer, Heidelberg, 2003.

[20] C. Billiet, J. Pons, T. Matthé, G. De Tré, and O. Pons, "Bipolar fuzzy querying of temporal databases," in *Lecture Notes in Artificial Intelligence*, vol. 7022, (Ghent, Belgium), pp. 60–71, Springer, Octobre 2011.

[21] D. Dubois, A. HadjAli, and H. Prade, "Fuzziness and uncertainty in temporal reasoning," *Journal of Universal Computer Science*, vol. 9, pp. 1168–1194, jan 2003.

[22] H. J. Ohlbach, "Relations between fuzzy time intervals," *International Symposium on Temporal Representation and Reasoning*, vol. 0, pp. 44–51, 2004.

[23] S. Schockaert, M. De Cock, and E. Kerre, "Fuzzifying allen's temporal interval relations," *Fuzzy Systems, IEEE Transactions on*, vol. 16, pp. 517 –533, april 2008.

[24] C. Garrido, N. Marin, and O. Pons, "Fuzzy intervals to represent fuzzy valid time in a temporal relational database," *Int. J. Uncertainty Fuzziness Knowlege-Based Syst.*, vol. 17, pp. 173–192, 2009.

[25] A. Bronselaer, J. E. Pons, G. De Tré, and O. Pons, "Possibilistic evaluation of sets," *Int. J. Uncertainty Fuzziness Knowlege-Based Syst.*, vol. 21, no. 3, pp. 325–346, 2013.

[26] Y. Qiang, K. Asmussen, M. Delafontaine, G. De Tré, B. Stichelbaut, P. De Maeyer, and N. Van de Weghe, "Visualising rough time intervals in a two-dimensional space," in *2009 IFSA World Congress / EUSFLAT Conference, Proceedings*, Jul 2009.

[27] Z. Pawlak, J. Grymala-Busse, R. Slowinski, and W. Ziarko, "Rough Sets," *Communications of the ACM*, vol. 38, no. 6, 1995.

[28] T. Imieliński and W. Lipski, Jr., "Incomplete information in relational databases," *J. ACM*, vol. 31, pp. 761–791, Sept. 1984.

[29] Y. Vassiliou, "Null values in data base management a denotational semantics approach," in *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, (New York, NY, USA), pp. 162–169, ACM, 1979.

[30] Y. Vassiliou, "Functional dependencies and incomplete information," in *Proc. 6th Int. Conf. on Very Large Data Bases* (N. Y. ACM, ed.), (Montreal, Ont. Canada), pp. 268–269, Oct 1980.

[31] J. Grant, "Null values in a relational data base," *Inf. Process. Lett.*, vol. 5, no. 5, pp. 156–157, 1977.

[32] C. Zaniolo, "Database relations with null values," in *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (A. N. York, ed.), (Los Angeles, Calif.), pp. 27–33, March 1982.

[33] G. D. Tré, R. M. M. D. Caluwe, and H. Prade, "Null values revisited in prospect of data integration," in *ICSNW* (M. Bouzeghoub, C. A. Goble, V. Kashyap, and S. Spaccapietra, eds.), vol. 3226 of *Lecture Notes in Computer Science*, pp. 79–90, Springer, 2004.

[34] A. Kolmogorov, *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Berlin: Julius Springer, 1933.

[35] B. D. Finetti, "La prévision : ses lois logiques, ses sources subjectives," *Ann. Inst. Poincaré*, vol. 7, pp. 1–68, 1937.

[36] Lotfi Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems*, vol. 1, pp. 3–28, 1978.

[37] Henri Prade, "Possibility sets, fuzzy sets and their relation to lukasiewicz logic," in *Proceedings of the International Symposium on Multiple-Valued Logic*, pp. 223–227, 1982.

[38] Didier Dubois and Henri Prade, *Possibility Theory*. Plenum Press, 1988.

[39] D. Dubois and H. Prade, "Interval-valued fuzzy sets, possibility theory and imprecise probability," in *In Proceedings of International Conference in Fuzzy Logic and Technology*, pp. 314–319, 2005.

[40] M. Zemankova-Leech and A. Kandel, "Fuzzy relational databases - a key to expert systems," *Journal of the American Society for Information Science*, vol. 37, pp. 272–273, 1984.

[41] B. P. Buckles and F. E. Petry, "Extending the fuzzy database with fuzzy numbers," *Information Sciences*, vol. 34, no. 2, pp. 145 – 155, 1984.

[42] B. P. Buckles and F. E. Petry, "A fuzzy representation of data for relational databases," *Fuzzy Sets and Systems*, vol. 7, no. 3, pp. 213 – 226, 1982.

[43] H. Prade and C. Testemale, "Representation of soft constraints and fuzzy attribute values by means of possibility distributions in data bases," *The Analysis of Fuzzy Information*, vol. II, 1987.

[44] H. Prade and C. Testemale, "Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries," *Information Sciences*, vol. 34, no. 2, pp. 115 – 143, 1984.

[45] E. A. Rundensteiner, L. W. Hawkes, and W. Bandler, "On nearness measures in fuzzy relational data models," *International Journal of Approximate Reasoning*, vol. 3, no. 3, pp. 267 – 298, 1989.

[46] S. Shenoi and A. Melton, "Proximity relations in the fuzzy relational database model," *Fuzzy Sets and Systems*, vol. 100, Supplement 1, no. 0, pp. 51 – 62, 1989.

[47] M. A. Vila, J. C. Cubero, J. M. Medina, and O. Pons, "A logic approach to fuzzy relational databases," *International Journal of Intelligent Systems*, vol. 9, pp. 449–460, 1994.

[48] G. de Tre, "Extended possibilistic truth values," *International Journal of Intelligent Systems*, vol. 17, pp. 427–446, 2002.

[49] P. Bosc, M. Galibourg, and G. Hamon, "Fuzzy querying with sql: extensions and implementation aspects," *Fuzzy Sets Syst.*, vol. 28, pp. 333–349, December 1988.

[50] J. Kacprzyk and A. Ziólkowski, "Database queries with fuzzy linguistic quantifiers," *IEEE Trans. Syst. Man Cybern.*, vol. 16, pp. 474–479, May 1986.

[51] J. Kacprzyk, S. Zadrożny, and A. Ziołkowski, "Fquery iii+: A "human-consistent" database querying system based on fuzzy logic with linguistic quantifiers," *Information Systems*, vol. 14, no. 6, pp. 443 – 453, 1989. ¡ce:title¿Fuzzy Databases¡/ce:title¿.

[52] G. De Tré and e. a. Zadrozny, "Dealing with Positive and Negative Query Criteria in Fuzzy Database Querying Bipolar Satisfaction Degrees," in *Proceedings of 8th Int. Conf. FQAS*, (Denmark), pp. 593–604, Springer Verlag Berlin, 2009.

[53] P. Bosc, O. Pivert, and K. Farquhar, "Integrating fuzzy queries into an existing database management system: An example," *International Journal of Intelligent Systems*, vol. 9, no. 5, pp. 475–492, 1994.

[54] J. Galindo, J. Medina, O. Pons, and J. Cubero, "A server for fuzzy sql queries," in *Flexible Query Answering Systems* (T. Andreasen, H. Christiansen, and H. Larsen, eds.), vol. 1495 of *Lecture Notes in Computer Science*, pp. 164–174, Springer Berlin / Heidelberg, 1998.

[55] M. Umano, "Freedom-0: A fuzzy database system.," *Fuzzy Inf and Decis Processes*, pp. 339–347, 1982. cited By (since 1996) 36.

[56] L. Tineo, M. Goncalves, and J. C. Eduardo, "A fuzzy querying system based on sqlf2 and sqlf3," in *CLEI2004* (M. Solar, D. Fernández-Baca, and E. Cuadros-Vargas, eds.), pp. 845–851, Sept. 2004.

[57] J. Galindo, *Fuzzy Databases: Modeling, Design, and Implementation*. Hershey, PA, USA: IGI Publishing, 2006.

[58] J. Clifford and A. U. Tansel, "On an algebra for historical relational databases: two views," *SIGMOD Rec.*, vol. 14, pp. 247–265, May 1985.

[59] M. R. Klopprogge and P. C. Lockemann, "Modelling information preserving databases: Consequences of the concept of time," in *Proceedings of the 9th International Conference on Very Large Data Bases*, (San Francisco, CA, USA), pp. 399–416, Morgan Kaufmann Publishers Inc., 1983.

[60] N. L. Sarda, "Extensions to sql for historical databases," *IEEE Trans. Knowl. Data Eng.*, vol. 2, pp. 220–230, 1990.

[61] R. Cavallo and M. Pittarelli, "The theory of probabilistic databases," in *Proc. 13th Int. Conf. Very Large Databases (VLDB'87)*, pp. 71–81, 1987.

[62] D. Barbara, H. Garcia-Molina, and D. Porter, "The management of probabilistic data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 4, pp. 487 –502, oct 1992.

[63] M. Pittarelli, "An algebra for probabilistic databases," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 6, pp. 293 –303, apr 1994.

[64] D. Dey and S. Sarkar, "A probabilistic relational model and algebra," *ACM Trans. Database Syst.*, vol. 21, pp. 339–369, Sept. 1996.

[65] L. Antova, C. Koch, and D. Olteanu, "Maybms: Managing incomplete information with probabilistic world-set decompositions," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 1479 –1480, april 2007.

[66] S. Abiteboul and P. Senellart, "Querying and updating probabilistic information in xml," in *Advances in Database Technology - EDBT 2006* (Y. Ioannidis, M. Scholl, J. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, eds.), vol. 3896 of *Lecture Notes in Computer Science*, pp. 1059–1068, Springer Berlin / Heidelberg, 2006. 10.1007/11687238_62.

[67] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian, "Probview: A flexible probabilistic database system," *ACM TRANSACTIONS ON DATABASE SYSTEMS*, vol. 22, no. 3, pp. 419–469, 1997.

[68] A. Nierman and H. V. Jagadish, "Protdb: probabilistic data in xml," in *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pp. 646–657, VLDB Endowment, 2002.

[69] M. van Keulen, A. de Keijzer, and W. Alink, "A probabilistic xml approach to data integration," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 459 – 470, april 2005.

[70] A. Dekhtyar, R. Ross, and V. S. Subrahmanian, "Probabilistic temporal databases, i: algebra," *ACM Trans. Database Syst.*, vol. 26, pp. 41–95, March 2001.

[71] A. Parker, V. S. Subrahmanian, and J. Grant, "A logical formulation of probabilistic spatial databases," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, pp. 1541–1556, Nov. 2007.

[72] F. Parisi, A. Parker, J. Grant, and V. S. Subrahmanian, "Scaling cautious selection in spatial probabilistic temporal databases," in *Methods for Handling Imperfect Spatial Information*, pp. 307–340, Springer, 2010.

[73] C. Dyreson and F. e. a. Grandi, "A consensus glossary of temporal database concepts," *SIGMOD Rec.*, vol. 23, pp. 52–64, 1994.

[74] C. E. Dyreson and R. T. Snodgrass, "Supporting valid-time indeterminacy," *ACM Trans. Database Syst.*, vol. 23, pp. 1–57, March 1998.

[75] W. Kurutach, "Modelling fuzzy interval-based temporal information: a temporal database perspective," in *Fuzzy Systems, 1995. International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE International Conference on*, vol. 2, pp. 741 –748 vol.2, mar 1995.

[76] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," *Theoretical Computer Science*, vol. 78, no. 1, pp. 159 – 187, 1991.

[77] P. Bosc and O. Pivert, "Modeling and querying uncertain relational databases: A survey of approaches based on the possible worlds semantics," *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, vol. 18, no. 5, pp. 565–603, 2010. cited By (since 1996) 2.

[78] P. Bosc and O. Pivert, "On a possibilistic database model with incomplete possibility distributions," in *Fuzzy Information Processing Society, 2009. NAFIPS 2009. Annual Meeting of the North American*, pp. 1 –6, june 2009.

[79] P. Bosc, O. Pivert, and H. Prade, "An uncertain database model and a query algebra based on possibilistic certainty," in *Soft Computing and Pattern Recognition (SoCPaR), 2010 International Conference of*, pp. 63 –68, dec. 2010.

[80] C. S. Jensen, R. T. Snodgrass, and M. D. Soo, "The tsql2 data model," in *The TSQL2 Temporal Query Language*, pp. 153–238, 1995.

[81] J. F. K. A. V. Benthem, *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse.* Reidel, Hingham, MA, 1982.

[82] N. A. Lorentzos, *A Formal Extension of the Relational Model for the Representation of Generic Intervals.* PhD thesis, Birkbeck College, University of London, 1988.

[83] C. S. Jensen and R. T. Snodgrass, "Temporal data management," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 36–44, 1999.

[84] M. Bohlen, R. Busatto, and C. Jensen, "Point-versus interval-based temporal data models," in *Data Engineering, 1998. Proceedings., 14th International Conference on*, pp. 192 –200, Feb. 1998.

[85] H. Lin, C. J. (codirector), M. Bohlen, R. Busatto, H. Gregersen, K. Torp, R. S. (codirector), A. Datta, and S. Ram, "Efficient conversion between temporal granularities," Master's thesis, The University of Arizona, 1997.

[86] X. S. Wang, S. Jajodia, and V. S. Subrahmanian, "Temporal modules: An approach toward federated temporal databases," in *INFORMATION SYSTEMS*, pp. 227–236, 1993.

[87] C. Dyreson and R. Snodgrass, "Temporal granularity and indeterminacy: Two sides of the same coin," technical report tr 94-06, Computer Science Department, University of Arizona, Tucson, U.S.A, Febuary 1994.

[88] S. Kraus, Y. Sagiv, and V. S. Subrahmanian, "Representing and integrating multiple calendars," tech. rep., University of Maryland at College Park, College Park, MD, USA, 1997.

[89] D. Husfeld and C. Kronberg, "Astronomical time keeping." `http://www.maa.mhn.de/Scholar/times.html`, 1996.

[90] "Converting between julian dates and gregorian calendar dates." http://www.usno.navy.mil/USNO/astronomical-applications/astronomical-information-center/julian-date-form/?searchterm=Julian

[91] D. Gambis, "The leap second." http://hpiers.obspm.fr/eop-pc/earthor/utc/leapsecond.html.

[92] "Leap second." http://en.wikipedia.org/wiki/Leap_second.

[93] H. F. Fliegel and T. C. van Flandern, "Letters to the editor: a machine algorithm for processing calendar dates," *Commun. ACM*, vol. 11, pp. 657–, October 1968.

[94] "Smithsonian astrophysical observatory." http://www.cfa.harvard.edu/sao/.

[95] "The official u.s time." http://nist.time.gov/.

[96] D. DuBois and H. Prade, "Processing fuzzy temporal knowledge," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, pp. 729–744, 1989.

[97] G. De Tré, R. De Caluwe, and B. Van der Cruyssen, "Dealing with time in fuzzy and uncertain object-oriented database models," *EUFIT'97*, pp. 1157–1161, Sep 1997.

[98] S. Dutta, "An event based fuzzy temporal logic," in *Multiple-Valued Logic, 1988., Proceedings of the Eighteenth International Symposium on*, pp. 64 –71, 0-0 1988.

[99] A. Motro, *Uncertainty Management in Information Systems from Needs to Solutions*, ch. Sources of Uncertainty, Imprecission and Inconsistency in Information Systems, pp. 2–27. Kluwer Academic Publishers, 1997.

[100] A. Motro, "Imprecision and uncertainty in database systems," *Fuzziness in Database Management Systems*, pp. 3–22, 1995.

[101] J. Virant and N. Zimic, "Attention to time in fuzzy logic," *Fuzzy Sets and Systems*, vol. 82, no. 1, pp. 39 – 49, 1996.

[102] P. Chountas and I. Petrounias, "Modelling and representation of uncertain temporal information," *Requirements Engineering*, vol. 5, pp. 144–156, 2000. 10.1007/s007660070006.

[103] F. Kabanza, J. m. Stevenne, and P. Wolper, "Handling infinite temporal data," in *Journal of Computer and System Sciences*, pp. 392–403, 1990.

[104] B. Knight and J. Ma, "Time representation: A taxonomy of temporal models," *Artificial Intelligence Review*, vol. 7, pp. 401–419, 1993. 10.1007/BF00849933.

[105] P. Haddawy, "Believing change and changing belief," *IEEE Transactions on Systems, Man, and Cybernetics Special Issue on Higher-Order Uncertainty*, vol. 26, 1996.

[106] G. Bordogna, P. Carrara, M. Pagani, M. Pepe, and A. Rampini, "Managing imperfect temporal metadata in the catalog services of spatial data infrastructures compliant with inspire.," in *IFSA/EUSFLAT Conf.* (J. P. Carvalho, D. Dubois, U. Kaymak, and J. M. da Costa Sousa, eds.), pp. 915–920, 2009.

[107] G. B. et al., *Advanced Database Query Systems*, ch. Flexible Querying of Imperfect Temporal Metadata in Spatial Data Infraestructures: Techniques, Applications and Technologies, p. 140. IGI Global, 2011.

[108] S. Soysangwarn and S. Chittayasothorn, "Toward fuzzy temporal databases with temporal fuzzy linguistic terms," in *Applications of Digital Information and Web Technologies, 2009. ICADIWT '09. Second International Conference on the*, pp. 8 –13, aug. 2009.

[109] A. Burney, N. Mahmood, and K. Ahsan, "Tempr-pdm: a conceptual temporal relational model for managing patient data," in *Proceedings of the 9th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases*, AIKED'10, (Stevens Point, Wisconsin, USA), pp. 237–243, World Scientific and Engineering Academy and Society (WSEAS), 2010.

[110] A. Burney, N. Mahmood, T. Jilani, and H. Saleem, "Conceptual fuzzy temporal relational model (ftrm) for patient data," *WSEAS Trans. Info. Sci. and App.*, vol. 7, pp. 725–734, May 2010.

[111] A. Burney, N. Mahmood, and Z. Abbas, "Advances in fuzzy rough set theory for temporal databases," in *Proceedings of the 11th WSEAS international conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, AIKED'12, (Stevens Point, Wisconsin, USA), pp. 237–242, World Scientific and Engineering Academy and Society (WSEAS), 2009.

[112] A. Bassiri, M. Malek, A. Alesheikh, and P. Amirian, "Temporal relationships between rough time intervals," in *Computational Science and Its Applications – ICCSA 2009* (O. Gervasi, D. Taniar, B. Murgante, A. Laganà, Y. Mun, and M. Gavrilova, eds.), vol. 5592 of *Lecture Notes in Computer Science*, pp. 543–552, Springer Berlin / Heidelberg, 2009.

[113] O. Etzion, S. Jajodia, and S. Sripada, *Temporal databases: research and practice*. Lecture notes in computer science, Springer, 1998.

[114] M. A. Nascimento and M. H. Eich, "Decision time in temporal databases," in *Proceedings of the Second International Workshop on Temporal Representation and Reasoning*, pp. 157–162, 1995.

[115] C. S. Jensen, L. Mark, and N. Roussopoulos, "Incremental implementation model for relational databases with transaction time," *IEEE Trans. Knowl. Data Eng.*, vol. 3, pp. 461–473, 1991.

[116] R. Snodgrass, "The temporal query language tquel," in *Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems*, PODS '84, (New York, NY, USA), pp. 204–213, ACM, 1984.

[117] Y. Wu, S. Jajodia, and X. Wang, "Temporal database bibliography update," in *Temporal Databases: Research and Practice* (O. Etzion, S. Jajodia, and S. Sripada, eds.), vol. 1399 of *Lecture Notes in Computer Science*, pp. 338–366, Springer Berlin / Heidelberg, 1998. 10.1007/BFb0053709.

[118] S. Navathe and R. Ahmed, "Tsql—a language interface for history databases," 5 1987.

[119] R. T. Snodgrass, C. Richard, T. Snodgrass, T. Snodgrass, C. S. Jensen, C. S. Jensen, C. S. Jensen, A. Steiner, A. Steiner, M. H. Böhlen, M. H. Böhlen, M. H. Böhlen, M. H. Böhlen, R. Busatto, H. Gregersen, C. S. J. (codirector, K. Torp, A. Datta, R. T. S. (codirector, and C. E. Dyreson, "Transitioning temporal support in tsql2 to sql3," in *Temporal Databases: Research and Practice* (O. Etzion, S. Jajodia, and S. Sripada, eds.), vol. 1399 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1998. 10.1007/BFb0053709.

[120] F.P.Brooks, *The Analytic Design of automatic data processing systems*. PhD thesis, Harvard University, 1956.

[121] R. L. Blum, "Displaying clinical data from a time-oriented database," *Comput. Biol. Med.*, vol. 11, pp. 197–210, January 1981.

[122] S. M. Joses and R.Stamper, "Legol 2.0: A relational specification language for complex rules," *Information Systems*, vol. 4, pp. 293– –305, 11 1979.

[123] J. Clifford, "A model for historical databases," *NYU Working paper*, 11 1982.

[124] J. Clifford and D. S. Warren, "Formal semantics for time in databases," *ACM Trans. Database Syst.*, vol. 8, pp. 214–254, June 1983.

[125] G. Ariav, "A temporally oriented data model," *ACM Trans. Database Syst.*, vol. 11, pp. 499–527, December 1986.

[126] R. Sadeghi, *A database query language for operations on historical data*. PhD thesis, 1987.

[127] A. Segev and A. Shoshani, "Logical modeling of temporal data," *SIGMOD Rec.*, vol. 16, pp. 454–466, December 1987.

[128] J. Clifford and A. Croker, "The historical relational data model (hrdm) and algebra based on lifespans," in *Proceedings of the Third International Conference on Data Engineering*, (Washington, DC, USA), pp. 528–537, IEEE Computer Society, 1987.

[129] A. U. Tansel, "Adding time dimension to relational model and extending relational algebra," *Inf. Syst.*, vol. 11, pp. 343–355, October 1986.

[130] S. K. Gadia and C.-S. Yeung, "A generalized model for a relational temporal database," *SIGMOD Rec.*, vol. 17, pp. 251–259, June 1988.

[131] S. K. Gadia and J. H. Vaishnav, "A query language for a homogeneous temporal database," in *Proceedings of the fourth ACM SIGACT-SIGMOD symposium on Principles of database systems*, PODS '85, (New York, NY, USA), pp. 51–56, ACM, 1985.

[132] S. K. Gadia, "Toward a multihomogeneous model for a temporal database," in *Proceedings of the Second International Conference on Data Engineering*, (Washington, DC, USA), pp. 390–397, IEEE Computer Society, 1986.

[133] B. Chuen-sing Yeung, *Query languages for a heterogeneous temporal database*. PhD thesis, Texas Tech University, 1986.

[134] N. Lorentzos and R. Johnson, "An extension of the relational model to support generic intervals," in *Advances in Database Technology—EDBT '88* (J. Schmidt, S. Ceri, and M. Missikoff, eds.), vol. 303 of *Lecture Notes in Computer Science*, pp. 528–542, Springer Berlin / Heidelberg, 1988. 10.1007/3-540-19074-0_71.

[135] N. A. Lorentzos and V. J. Kollias, "The handling of depth and time intervals in soil-information systems," *Comput. Geosci.*, vol. 15, pp. 395–401, March 1989.

[136] K. Kimball, "The data system," Master's thesis, Pennsylvania, 1978.

[137] L. A. Rowe and M. Stonebraker, *The Postgres Papers*. Berkeley, CA, USA: University of California at Berkeley, 1987.

[138] N. A. Lorentzos and Y. G. Mitsopoulos, "Sql extension for interval data," *IEEE Trans. on Knowl. and Data Eng.*, vol. 9, pp. 480–499, May 1997.

[139] M. H. Böhlen and C. S. Jensen, "Seamless integration of time into sql," tech. rep., Aalborg University, 1996.

[140] A. Steiner, *A Generalisation Approach to Temporal Data Models and their Implementation*. PhD thesis, Swiss federal institute of technology, Zurich, 1998.

[141] R. Snodgrass, I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada., "Tsql2 language specification," tech. rep., University of Arizona, Tucson, September 1994.

[142] S. K. Gadia, "A homogeneous relational model and query languages for temporal databases," *ACM Trans. Database Syst.*, vol. 13, pp. 418–448, October 1988.

[143] D. Toman, "A point-based temporal extension of sql," in *In International Conference on Deductive and Object-Oriented Databases*, pp. 103–121, 1997.

[144] Oracle Corp., *Oracle database 11g Workspace Manager Overview*, 09 2009.

[145] TimeDB, "A temporal relational dbms.," 12 2011.

[146] Postgree, "Temporal postgreesql," 12 2011.

[147] Teradata, "Teradata temporal," 12 2011.

[148] S. Dieker and R. H. Güting, "Plug and play with query algebras: Secondo a generic dbms development environment," in *Proceedings of the 2000 International Symposium on Database Engineering & Applications*, IDEAS '00, (Washington, DC, USA), pp. 380–392, IEEE Computer Society, 2000.

[149] R. H. Güting and M. Schneider, "Moving objects databases."

[150] D. Dubois and H. Prade, "Ranking fuzzy Numbers in the Setting of Possibility Theory," *Information Sciences*, vol. 224, pp. 183–224, 1983.

[151] D. Dubois and H. Prade, *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. New York: Plenum Press, 1988.

[152] Didier Dubois and Henri Prade, "The three semantics of fuzzy sets," *Fuzzy Sets and Systems*, vol. 90, no. 2, pp. 141–150, 1997.

[153] P. Krause and D. Clark, "Representing uncertain knowledge," 1993.

[154] D. Dubois and H. Prade, "Incomplete conjunctive information," *Computers & Mathematics with Applications*, vol. 15, pp. 797–810, 1988.

[155] Didier Dubois and Henri Prade, *Decision-making Process: Concepts and Methods*, ch. Formal representations of uncertainty, pp. 85–156. Wiley, 2009.

[156] G. Shafer, *A mathematical theory of evidence*. Princeton University Press, 1961.

[157] J. D. Ullman, *Principles of Database Systems*. Computer Science Press, 1982.

[158] M. Umano and S. Fukami, "Retrieval processing from fuzzy databases," *Technical Reports of IECE of Japan*, vol. 80, no. 204, pp. 45 – 54, 1980.

[159] J. Medina, O. Pons, and J. Cubero, "Gefred. a generalized model of fuzzy relational databases," *Information Sciences*, vol. 76, pp. 87–109, 1994.

[160] J. Galindo, "New characteristics in fsql, a fuzzy sql for fuzzy databases," in *Proceedings of the 4th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering Data Bases*, AIKED'05, (Stevens Point, Wisconsin, USA), pp. 4:1–4:9, World Scientific and Engineering Academy and Society (WSEAS), 2005.

[161] H. Prade, "Current research trends in possibilistic logic: Multiple agent reasoning, preference representation, and uncertain databases," in *Advances in Data Management* (Z. Ras and A. Dardzinska, eds.), vol. 223 of *Studies in Computational Intelligence*, pp. 311–330, Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-02190-9_15.

[162] L. Zadeh, "Similarity relations and fuzzy orderings," *Information Sciences*, vol. 3, no. 2, pp. 177 – 200, 1971.

[163] M. Anvari and G. F. Rose, "Fuzzy relational databases," *Analysis of Fuzzy Information*, vol. II, pp. 203–212, 1987.

[164] B. Buckles, F. Petry, and H. Sachar, "A domain calculus for fuzzy relational databases," *Fuzzy Sets Syst.*, vol. 29, pp. 327–340, 1989.

[165] S. Shenoi and A. Melton, "An extended version of the fuzzy relational database model," *Information Sciences*, vol. 51, pp. 35–52, 1990.

[166] M. Umano, *Fuzzy Information, Knowledge Representation and Decision Analysis*, ch. Retrieval from Fuzzy Database by Fuzzy Relational Algebra, pp. 1–6. Pergamon Press, New York, 1983.

[167] M. Umano and S. Fukami, "Fuzzy relational algebra for possibility-distribution-fuzzy-relational model of fuzzy data," *Journal of Intelligent Information Systems*, vol. 3, pp. 7–28, 1994.

[168] M. Zemankova-Leech and A. Kandel, "Implementing imprecision in information systems," *Information Sciences*, vol. 37, pp. 107–141, 1985.

[169] R. De Caluwe, F. Devos, P. Maesfranckx, G. De Tré, and B. Van Der Cruyssen, "The semantics and modelling of flexible time indications.," *Computing with Words in Information / Intelligent Systems*, vol. 1: Foundations, pp. 229–256, 1999.

[170] R. De Caluwe, G. De Tré, B. Van Der Cruyssen, F. Devos, and P. Maesfranckx, *Time management in fuzzy and uncertain object-oriented databases*, vol. 39 of *Knowledge management in fuzzy databases*, pp. 67–88. Physica-Verlag, 2000.

[171] Brian Gaines and Ladislav Kohout, "Possible automata," in *Proceedings of the International symposium on multiple-valued logic*, (Bloomington, USA), pp. 183–196, 1975.

[172] Gert De Cooman, "Possibility theory 1: The measure- and integral-theoretic groundwork," *International Journal of General Systems*, vol. 25, no. 4, pp. 291–323, 1997.

[173] Gert De Cooman, "Possibility theory 2: Conditional possibility," *International Journal of General Systems*, vol. 25, no. 4, pp. 325–351, 1997.

[174] Gert De Cooman, "Possibility theory 3: Possibilistic independence," *International Journal of General Systems*, vol. 25, no. 4, pp. 353–371, 1997.

[175] D. Dubois and H. Prade, "Flexible query answering systems," ch. Using fuzzy sets in flexible querying: why and how?, pp. 45–60, Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[176] G. Choquet, "Theory of capacities," *Annales de l'Institut Fourier*, vol. 5, pp. 131–295, 1953.

[177] D. Dubois and H. Prade, "Ranking fuzzy numbers in the setting of possibility theory," *Information Sciences*, vol. 30, pp. 183–224, 1983.

[178] E. McCluskey, *Introduction to the Theory of Switching Circuits*. McGraw-Hill Book Company, 1965.

[179] J. Galindo and J. M. Medina, "Ftsql2: Fuzzy time in relational databases," in *EUSFLAT Conf. '01*, pp. 47–50, 2001.

[180] D. Dubois, A. HadjAli, and H. Prade, "Fuzziness and uncertainty in temporal reasoning," *j-jucs*, vol. 9, pp. 1168–1194, 1 2003.

[181] J. E. Pons, C. Billiet, O. Pons, and G. de Tré, "A possibilistic valid-time model," in *Proceedings of the 14th International Conference on Information Processing and Management of Uncertainty on Knowledge-Based Systems*, 2012.

[182] J. Pons, A. Bronselaer, O. Pons, and G. de Tre, "Possibilistic evaluation of fuzzy temporal intervals," (Valladolid, Spain), february 2012.

[183] C. S. Jensen, R. T. Snodgrass, and M. D. Soo, "The tsql2 data model," 1994.

[184] D. Gao, C. S. Jensen, R. T. Snodgrass, and M. D. Soo, "Join operations in temporal databases," tech. rep., TimeCenter, 2002.

[185] V. Tahani, "A conceptual framework for fuzzy query processing - a step toward very intelligent database systems," *Inf. Process. Manage.*, vol. 13, no. 5, pp. 289–303, 1977.

[186] P. Bosc and O. Pivert, "An approach for a hierarchical aggregation of fuzzy predicates," in *Fuzzy Systems, 1993., Second IEEE International Conference on*, pp. 1231 –1236 vol.2, 1993.

[187] D. Dubois and H. Prade, "Bipolarity in flexible querying," in *Proceedings of the 5th International Conference on Flexible Query Answering Systems*, FQAS '02, (London, UK, UK), pp. 174–182, Springer-Verlag, 2002.

[188] D. Dubois and H. Prade, *Handbook of Research on Fuzzy Information Processing in Databases*, ch. Handling bipolar queries in Fuzzy Information Processing, pp. 97–114. New York, USA: Information Science Reference, 2008.

[189] P. Bosc, O. Pivert, A. Mokhtari, and L. Liétard, "Extending relational algebra to handle bipolarity," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (New York, NY, USA), pp. 1718–1722, ACM, 2010.

[190] L. Lietard, D. Rocacher, and P. Bosc, "On the extension of sql to fuzzy bipolar conditions," in *Fuzzy Information Processing Society, 2009. NAFIPS 2009. Annual Meeting of the North American*, pp. 1 –6, june 2009.

[191] L. Lietard and D. Rocacher, "On the Definition of Extended Norms and Conorms to Aggregate Fuzzy Bipolar Conditions," in *European Society for Fuzzy Logic and Technology*, pp. 513–518, 2009.

[192] L. Liétard, N. Tamani, and D. Rocacher, "Linguistic quantifiers and bipolarity.," in *Proc. of the 2011 IFSA World Congress and the 2011 AFSS International Conference.*, (Surabaya and Bali Island, Indonesia), 2011.

[193] N. Tamani, L. Liétard, and D. Rocacher, "Bipolarity and the relational division," in *Proc. of the 7th conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2011).*, (Aix-les-Bains France), 2011.

[194] S. Zadrozny and J. Kacprzyk, "Bipolar queries and queries with preferences (invited paper)," in *Database and Expert Systems Applications, 2006. DEXA '06. 17th International Workshop on*, pp. 415 –419, 0-0 2006.

[195] S. Zadrozny, G. D. Tre, and J. Kacprzyk, "Remarks on various aspects of bipolarity in database querying," *2012 23rd International Workshop on Database and Expert Systems Applications*, vol. 0, pp. 323–327, 2010.

[196] T. Matthé and G. De Tré, "Bipolar query satisfaction using satisfaction and dissatisfaction degrees: Bipolar satisfaction degrees," in *Proc. of the ACM SAC'09 Conference*, (Honolulu, Hawaii, USA), pp. 1699–1703, 2009.

[197] T. Matthé, G. De Tré, S. Zadrozny, J. Kacprzyk, and A. Bronselaer, "Bipolar database querying using bipolar satisfaction degrees," *Int. J. Intell. Syst.*, vol. 26, pp. 890–910, Oct. 2011.

[198] M. Lacroix and P. Lavency, "Preferences; putting more knowledge into queries," in *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, (San Francisco, CA, USA), pp. 217–225, Morgan Kaufmann Publishers Inc., 1987.

[199] G. De Tré, R. D. Caluwe, J. Kacprzyk, and S. Zadrozny, "On flexible database querying via extensions to fuzzy sets," in *Proc. of Joint EUSFLAT-LFA 2005*, 2005.

[200] P. Bosc and O. Pivert, "Some approaches for relational databases flexible querying," *Journal of Intelligent Information Systems*, vol. 1, pp. 323–354, 1992.

[201] P. Bosc and O. Pivert, "Sqlf: a relational database language for fuzzy querying," *Fuzzy Systems, IEEE Transactions on*, vol. 3, pp. 1 –17, feb 1995.

[202] J. Kacprzyk and S. Zadrozny, *Fuzziness in database management systems*, ch. FQUERY for Access: Fuzzy querying for windows-based DBMS, pp. 415–433. Physica-Verlag, 1995.

[203] P. Bosc, D. Kraft, and F. Petry, "Fuzzy sets in database and information systems: Status and opportunities," *Fuzzy Sets and Systems*, vol. 156, no. 3, pp. 418 – 426, 2005. 40th Anniversary of Fuzzy Sets.

[204] S. Zadrozny, G. de Tré, R. de Caluwe, and J. Kacprzyk, *Handbook of Research on Fuzzy Information Processing in Databases*, ch. An overview of fuzzy approaches to flexible database querying, pp. 34–54. New York, USA: Information Science Reference, 2008.

[205] M. De Calmès, D. Dubois, E. Hüllermeier, H. Prade, and F. Sèdes, "A fuzzy set approach to flexible case-based querying: methodology and experimentation," in *Proc. of the 8th International Conference, Principles of Knowledge Representation and Reasoning (KR2002)*, (Toulouse, France), pp. 449–458, 2002.

[206] K.-U. Jahn, "Intervall-wertige mengen," *Mathematische Nachrichten*, vol. 68, no. 1, pp. 115–132, 1975.

[207] R. Sambuc, *Fonctions ϕ-floues. Application l'aide au diagnostic en pathologie thyroidienne*. PhD thesis, Univ. Marseille, France, 1975.

[208] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning - i," *Inf. Sci.*, vol. 8, no. 3, pp. 199–249, 1975.

[209] I. Grattan-Guinness, "Fuzzy membership mapped onto intervals and many-valued quantities," *Mathematical Logic Quarterly*, vol. 22, no. 1, pp. 149–160, 1976.

[210] K. T. Atanassov, "Intuitionistic fuzzy sets," *Fuzzy Sets and Systems*, vol. 20, pp. 87–96, 1986.

[211] F. Smarandache, *A Unifying Field in Logics: Neutrosophic Logic. Neutrosophy, Neutrosophic Set, Neutrosophic Probability.* American Research Press, 1999.

[212] U. Rivieccio, "Neutrosophic logics: Prospects and problems," *Fuzzy Sets Syst.*, vol. 159, pp. 1860–1868, July 2008.

[213] N. D. Belnap, *Modern Uses of Multiple-Valued Logic*, ch. A useful four-valued logic, pp. 8–37. Dordrecht, The Netherlands: Reidel, 1977.

[214] M. Öztürk and A. Tsoukiàs, "Modelling uncertain positive and negative reasons in decision aiding," *Decision Support Systems*, vol. 43, no. 4, pp. 1512–1526, 2007.

[215] E. Turunen, M. Öztürk, and A. Tsoukiàs, "Paraconsistent semantics for Pavelka style fuzzy sentential logic," *Fuzzy Sets and Systems*, vol. 161, no. 14, pp. 1926–1940, 2010.

[216] S. Konieczny, P. Marquis, and P. Besnard, "Bipolarity in bilattice logics," *International Journal of Intelligent Systems*, vol. 23, no. 10, pp. 1046–1061, 2008.

[217] G. Bordogna and G. Pasi, "Linguistic aggregation operators of selection criteria in fuzzy information retrieval," *International Journal of Intelligent Systems*, vol. 10, no. 2, pp. 233–248, 1995.

[218] D. Dubois and H. Prade, "Default reasoning and possibility theory," *Artificial Intelligence*, vol. 35, no. 2, pp. 243–257, 1988.

[219] R. Yager, "Fuzzy logic in the formulation of decision functions from linguistic specifications," *Kybernetes*, vol. 25, no. 4, pp. 119–130, 1996.

[220] P. Bosc and O. Pivert, "On three fuzzy connectives for flexible data retrieval and their axiomatization," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, (New York, NY, USA), pp. 1114–1118, ACM, 2011.

[221] L. Lietard, N. Tamani, and D. Rocacher, "Fuzzy bipolar conditions of type or else," in *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, pp. 2546 –2551, june 2011.

[222] P. Bosc and O. Pivert, "On four noncommutative fuzzy connectives and their axiomatization," *Fuzzy Sets and Systems*, vol. 202, no. 0, pp. 42 – 60, 2012. ¡ce:title¿Theme: Aggregation Functions¡/ce:title¿.

[223] T. Matthé and G. Tré, "Ranking of bipolar satisfaction degrees," in *Advances in Computational Intelligence* (S. Greco, B. Bouchon-Meunier, G. Coletti, M. Fedrizzi, B. Matarazzo, and R. Yager, eds.), vol. 298 of *Communications in Computer and Information Science*, pp. 461–470, Springer Berlin Heidelberg, 2012.

[224] T. Matthé and G. de Tré, "Weighted aggregation of bipolar satisfaction degrees.," in *Proc. of the 2011 IFSA World Congress and the 2011 AFSS International Conference.*, 2011.

[225] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," *IEEE Trans. Syst. Man Cybern.*, vol. 18, pp. 183–190, Jan. 1988.

[226] R. R. Yager and J. Kacprzyk, eds., *The ordered weighted averaging operators: theory and applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[227] J. Pons, N. Marín, O. Pons, C. Billiet, and G. de Tré, "A relational model for the possibilistic valid-time approach," *International Journal of Computational Intelligence Systems*, vol. 5, no. 6, pp. 1068–1088, 2012.

[228] J. Deploige, B. Callens, P. Demonty, and G. De Tré, "Remedying the obsolescence of digitised surveys of medieval sources. narrative sources and diplomata belgica."

[229] C. Garrido, N. Marin, and O. Pons, "Fuzzy intervals to represent fuzzy valid time in a temporal relational database," *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, vol. 17, no. SUPPL. 1, pp. 173–192, 2009.

[230] C. Billiet, J. Pons, O. Pons, and G. Tré, "Evaluating possibilistic valid-time queries," in *Advances on Computational Intelligence* (S. Greco, B. Bouchon-Meunier, G. Coletti, M. Fedrizzi, B. Matarazzo, and R. Yager, eds.), vol. 297 of *Communications in Computer and Information Science*, pp. 410–419, Springer Berlin Heidelberg, 2012.

[231] Z. Kulpa, "Diagrammatic representation of interval space in proving theorems about interval relations," *Reliable Computing*, vol. 3, pp. 209–217, 1997.

[232] Z. Kulpa, "A diagrammatic approach to investigate interval relations," *Journal of Visual Languages & Computing*, vol. 17, no. 5, pp. 466 – 502, 2006. Context and Emotion Aware Visual Interaction - Part II.

[233] N. V. de Weghe, R. Docter, P. D. Maeyer, B. Bechtold, and K. Ryckbosch, "The triangular model as an instrument for visualising and analysing residuality," *Journal of Archaeological Science*, vol. 34, no. 4, pp. 649 – 655, 2007.

[234] G. Tré, N. Weghe, R. Caluwe, and P. Maeyer, "Towards a flexible visualization tool for dealing with temporal data," in *Flexible Query Answering Systems* (H. Larsen, G. Pasi, D. Ortiz-Arroyo, T. Andreasen, and H. Christiansen, eds.), vol. 4027 of *Lecture Notes in Computer Science*, pp. 109–120, Springer Berlin Heidelberg, 2006.

[235] Y. Qiang, M. Delafontaine, K. Asmussen, B. Stichelbaut, G. de Tré, P. D. Maeyer, and N. V. de Weghe, "Modelling imperfect time intervals in a two-dimensional space," *Control and Cybernetics*, vol. 39, no. 4, pp. 983–1010, 2010.

[236] Y. Qiang, *Modelling Temporal Information in a Two-Dimensional Space: A visualization perspective.* PhD thesis, Ghent University, Ghent, Belgium, 2012.

[237] G. King, C. Bauer, M. R. Andersen, E. Bernard, S. Ebersole, and H. Ferentschik, *Hibernate Reference Documentation*, 3.6.0.cr2 ed. `http://www.hibernate.org/docs`.

[238] O. Corp., "Java se technologies - database." `http://www.oracle.com/technetwork/java/javase/jdbc/index.html`. Accessed Nov. 2012.

[239] O. Corp., "Java transaction api (jta)." http://www.oracle.com/technetwork/java/javaee/jta/index.html. Accessed Nov. 2012.

[240] "Javabeans." http://en.wikipedia.org/wiki/JavaBeans. Accessed on Nov. 2012.

[241] "Plain old java object." http://en.wikipedia.org/wiki/Plain_Old_Java_Object. Accessed on Nov. 2012.

[242] J. Org., "Working with objects.." http://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html_single/\#objectstate. Accessed on Nov. 2012.

[243] I. Object Management Group, "Corba specification." http://www.omg.org/corba/. Accessed on Nov.2012.

[244] "Common object request broker architecture." http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture. Accessed on Nov. 2012.

[245] wikipedia, "Enterprise javabeans." http://en.wikipedia.org/wiki/Enterprise_JavaBeans.

[246] J. Comunity, "Enterprise java beans (ejb) 3.0." http://www.jboss.org/ejb3.

[247] Hibernate, "Fetching strategies." http://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html_single/\#performance-fetching.

[248] Hibernate, "Hql: The hibernate query language." http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html.

[249] O. Corp., "Using the criteria api to create queries." http://docs.oracle.com/javaee/6/tutorial/doc/gjitv.html.

[250] Wikipedia, "Query by example." http://en.wikipedia.org/wiki/Query_by_Example.

[251] K. Torp, C. Jensen, and R. Snodgrass, "Stratum approaches to temporal dbms implementation," in *Database Engineering and Applications Symposium, 1998. Proceedings. IDEAS'98. International*, pp. 4 –13, jul 1998.

[252] C. Martínez Cruz, *Sistema de gestión de bases de datos relacionales difusas multipropósito.* PhD thesis, Universidad de Granada, 2008.

[253] H. Nakajima, T. Sogoh, and M. Arao, "Fuzzy database language and library-fuzzy extension to sql," in *Fuzzy Systems, 1993.*, pp. 477 –482 vol.1, 1993.

[254] S. Škrbic and A. Takaci, "An interpreter for priority fuzzy logic enriched sql," in *BCI '09*, (Washington, DC, USA), pp. 96–100, IEEE Computer Society, 2009.

[255] Antwan Van Schooten, *Ontwerp en implementatie van een model voor de representatie en manipulatie van onzekerheid en imprecisie in databanken en expert systemen*. PhD thesis, Ghent University, 1988.

[256] Gert De Cooman, "Towards a possibilistic logic. in: Fuzzy set theory and advanced mathematical applications, edited by da ruan, kluwer academic, pp. 89–133, boston.," 1995.