Abstract

Title of dissertation: Personalizable Knowledge Integration

Maria Vanina Martinez,
Doctor of Philosophy, 2011

Dissertation directed by: Professor V.S. Subrahmanian
Department of Computer Science

Large repositories of data are used daily as knowledge bases (KBs) feeding computer systems that support decision making processes, such as in medical or financial applications. Unfortunately, the larger a KB is, the harder it is to ensure its consistency and completeness. The problem of handling KBs of this kind has been studied in the AI and databases communities, but most approaches focus on computing answers locally to the KB, assuming there is some single, epistemically correct solution. It is important to recognize that for some applications, as part of the decision making process, users consider far more knowledge than that which is contained in the knowledge base, and that sometimes inconsistent data may help in directing reasoning; for instance, inconsistency in taxpayer records can serve as evidence of a possible fraud. Thus, the handling of this type of data needs to be context-sensitive, creating a synergy with the user in order to build useful, flexible data management systems.

Inconsistent and incomplete information is ubiquitous and presents a substantial problem when trying to reason about the data: how can we derive an adequate model of the world, from the point of view of a given user, from a KB that may be inconsistent or incomplete? In this thesis we argue that in many cases users need to bring their

application-specific knowledge to bear in order to inform the data management process. Therefore, we provide different approaches to handle, in a personalized fashion, some of the most common issues that arise in knowledge management. Specifically, we focus on (1) inconsistency management in relational databases, general knowledge bases, and a special kind of knowledge base designed for news reports; (2) management of incomplete information in the form of different types of null values; and (3) answering queries in the presence of uncertain schema matchings. We allow users to define policies to manage both inconsistent and incomplete information in their application in a way that takes both the user's knowledge of his problem, and his attitude to error/risk, into account. Using the frameworks and tools proposed here, users can specify when and how they want to manage/solve the issues that arise due to inconsistency and incompleteness in their data, in the way that best suits their needs.

PERSONALIZABLE
KNOWLEDGE INTEGRATION

by

Maria Vanina Martinez

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:

Professor V.S. Subrahmanian, Chair/Advisor
Professor John Grant
Professor Sarit Kraus
Professor Dana Nau
Professor Jonathan Wilkenfeld

Dedication

To: Gerardo, Cristina, Hector, Roman, and Mauricio.

Acknowledgements

In the first place, I would like to thank my family for having been there unconditionally for me throughout my studies. To my husband Gerardo, who supported and encouraged me to keep going forward in my good and bad moments, especially when being so far from my family was sometimes too hard. I am most grateful to my beloved parents, Cristina and Hector, and my brothers Mauricio and Roman. They were with me all this time, even in the distance, missing me very much, but always encouraging me to do what I wanted to do. They taught me the importance of learning and seeking knowledge, and that an education is one of the most important things that a person can have, independently of what path is taken in life. To them I owe who I am and what I have done so far. To the rest of my family: Gabriela, Guillermo, Amalia, and Patricio, who have also been there for me, and to my little nephew Sebastian.

I want to thank my advisor, Professor V.S. Subrahmanian, for giving me the opportunity to work and learn from him, and for giving me the opportunity of joining his group even before officially starting the program. His expertise as a researcher in the area and his seemingly endless generation of ideas was crucial for me to start building my career. He taught me to be a good researcher, and I was lucky to have him as an advisor. Thanks are also due to Professors John Grant, Sarit Kraus, Dana Nau, and Jonathan Wilkenfeld for agreeing to serve on my thesis committee, which meant putting valuable time aside to review the manuscript and provide feedback. Also to Professors John Grant, Avigdor Gal, and Sarit Kraus, from which I have learned so much by working with them.

During my time at UMD, I had the opportunity to meet many great people. some of them I collaborated with, and some became very good friends. My thanks then to Amy Sliva, Cristian Molinaro, Andrea Pugliese, Francesco Parisi, Massimiliano Albanese, Matthias Bröcheler, Paulo Shakarian, John Dickerson, and especially to Carlos Castillo and Gabriela Chavez, who became very good friends and with whom my husband and I spent so many good times.

Finally, I want to thank my friends in Argentina, the rest of my family there, and those who supported my decision of leaving my country to embark on this great adventure that was getting my PhD.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Knowledge Integration: Real World Application Issues

The process of knowledge integration involves combining knowledge bases residing in different sources and providing users with a *unified view* of these data. Researchers both in AI and databases, as well as in information retrieval, have been working on the problems that arise with the integration of heterogeneous knowledge bases for decades. The area of data integration is vast, and combines efforts from separate but related areas of computer science. Work in this area has focused on many issues at different levels of information integration during the last 30 years starting shortly after the global adoption of relational databases, which naturally led to the need for sharing and/or merging existing data repositories. Nowadays, solving problems associated with data integration applications is even more necessary given the information globalization produced by the WWW. A huge number of data sets are published and shared over the Internet daily, increasing the need for efficient and relatively accurate data integration tools.

The two main problems that arise when merging knowledge from different sources are *uncertainty* and *inconsistency*. In data integration applications, uncertainty can appear in different ways, but all of them can be traced to the subjectivity with which knowledge bases are built: knowledge bases model the knowledge about the world, *e.g.*, objects, their relevant properties, and relations among them, based on the requirements of the applications the were designed to serve. The same domain may be described in different, sometimes conflicting, ways. Differences may arise in the particular aspects of the domain they model, in the schemas designed for the domain (the structure of tables or relations, attribute names, and types), and in the naming conventions used for data objects. Inconsistency appears also as a result of integration. On one side, different sources of knowledge, which may be consistent in isolation, can each contain information that contradict each other when they are considered together. On the other side, since many data integration applications are based on structures that are automatically extracted from unstructured data, there may even be uncertainty about the data itself, since the extraction techniques are approximate at best. As data integration applications strive to offer a single objective and coherent integrated view of data sources, both uncertainty and inconsistency are bound to appear. These problems are pervasive especially in data integration applications whose goal is to offer access to a large number of heterogeneous sources, for instance when providing online services over the WWW. An everyday example of such a service is a search engine serving consumer travel needs (flights, rental cars, hotels, tourist packages, travel insurance, cruises, etc.); since different companies feed their data to such a service the problems discussed above seem inevitable.

Furthermore, large repositories of data are readily accessible and used daily as knowledge bases (KBs) that feed a wide variety of computer systems. One important type of systems are those that support decision making processes, *i.e.*, human users use them to interpret the data contained in a KB in order to make real world decisions. Examples of these applications are found in many environments such as medical, financial, cultural behavioral analysis, crime prevention, etc. Unfortunately, the larger a KB is, the harder it is to ensure its consistency and completeness. Inconsistent and incomplete (or partial) information is ubiquitous and presents a substantial problem when trying to reason about the data: how can we derive an adequate model of the world from a KB that we know may be inconsistent or incomplete? This situation can arise in the real world when maintaining, for instance, a KB with information about vehicle position and status, such as in robotics where Unmanned Ground Vehicles (UGVs) sense and act in an environment. For many reasons, such as unreliable communication channels or redundancy, there can be contradictory pieces of information regarding a vehicle in a certain instant of time. Depending on how a data management system handles this inconsistency, the user of this KB may have different options at hand when he comes into contact with the data. In general, these applications are managed by *power users*: users that utilize advanced features of programs which are beyond the abilities of "normal" or "end" users, but they are not necessarily capable of programming or familiar with how database management systems work. An important aspect to take into account in these applications is that, as part of the decision making process, users consider far more knowledge than that which is contained in the KB; they also incorporate into the process their domain expertise and requirements, as well as their expectations.

The management of uncertain knowledge bases has been widely studied both in the AI community and in the database community. Extensive work has been done in AI regarding uncertain information for a long time now, especially in the areas of nonmonotonic reasoning [Rei80a, MD80, McC87, Gab85, Moo88, KLM90], and the various topics studied in probabilistic reasoning [Hai84, Nil86b, Pea88, FHM90, KTG92, NS92, Poo93, FH94, Poo97, KIL04]. During the late 80's and 90's, proposals were made in the database community to incorporate probabilities into deductive and relational databases [CP87, BGMP92, LS94, NS94, LLRS97, FR97], each of them making different dependency assumptions with respect to probabilities. More recently, the database community has regained interest in probabilistic approaches, particularly in the area of query answering [AFM06, DS07, CKP03, BDSH^{+}08] and top-k querying [LCcI^{+}04, RDS07, SI07]. There has also been interest in uncertainty produced by the presence of *null values* or incomplete information [Gra80, IL84b, GJ86]. In this thesis we will focus on particular aspects of uncertainty related to inconsistency, schema matching, and incomplete information.

The problem of identifying and solving inconsistency in knowledge bases has been studied for many years by many different researchers [Gra78, BKM91, BDP97, BS98, ABC99, BFFR05, CLR03, BC03, Cho07]. Traditionally, the artificial intelligence (AI) and database theory communities held the posture that knowledge bases and software specifications should be completely free of inconsistent and incomplete information, and that inconsistency and incompleteness should be eradicated from them immediately. In the last two decades, however, these communities have recognized that for many interesting applications this posture is obsolete: Though approaches to allow inconsistency to persist in relational DBs and KBs have existed since the late 80s ([BS98, KS92, KL92, GS95, BKM91], etc.), there has been no method to date that gives the user the power to bring his knowledge of the domain, his preferences, and his risks and objectives into account when reasoning about inconsistent data. In this thesis we argue that inconsistency can often be resolved in different ways based on what the user wants. In the case of the vehicle KB above, a data management system that ignores the inconsistency and gives an "a priori" solution for it may hide the inconsistency from the user; this can be a problem if it causes the user to make the wrong decision and, for instance, delay the sending of rescue or support to disabled vehicles or to send it to the wrong location. Furthermore, contradictory information can be used in detecting faulty sensors or communication channels.

Consider now a simpler database example, a database containing data about employees in a company. We will use it to show the importance of giving the user the power to define and control the uncertainty in his data.

	Name	*Salary*	*Tax_bracket*	*Source*
t_1	$John$	$70K$	15	s_1
t_2	$John$	$80K$	20	s_2
t_3	$John$	$70K$	25	s_3
t_4	$Mary$	$90K$	30	s_1

Let us assume that salaries are uniquely determined by names, which means that for every two records in the database that have the exact same name, they should also have the exact same amount for salary. Clearly, there is an inconsistency regarding employee John in the table above. In this case, a user may want to resolve the inconsistency about John's salary in many different ways. (C1) If he were considering John for a loan, he might want to choose the lowest possible salary of John to base his loan on. (C2) If he were assessing the amount of taxes John has to pay, he may choose the highest possible salary John may have. (C3) If he were just trying to estimate John's salary, he may choose some number between 70K and 80K (*e.g.*, the average of the three reports of John's salary) as the number. (C4) if he had different degrees of confidence in the sources that provided these salaries, he might choose a weighted mean of these salaries. (C5) He might choose not to resolve the inconsistency at all, but to just let it persist until he can clear it up. (C6) He might simply consider *all* the data about John unreliable and might want to ignore it until it can be cleared up – this is the philosophy of throwing away all contaminated data.[1] [BKM91, SA07, ABC99, BFFR05, CLR03, BC03, Cho07] can handle cases C1 and C2, but not the other cases.

1.2 Organization of this Thesis

In this thesis we propose to provide users with tools to manage their data in a personalized way in order to reason about it according to their needs. Given that *it is important to enable users to bring their application-specific knowledge to bear when resolving inconsistency*, we propose two different approaches to personalizable inconsistency management: *Inconsistency Management Policies* for relational databases and a *general framework* for handling inconsistent knowledge bases. For the first approach we define the concept of a *policy* for managing inconsistency in relational databases with respect to functional dependencies, which generalizes other efforts in the database community by allowing policies to either remove inconsistency completely or to allow part or all of the inconsistency to persist depending on the users' application needs. In the example above, each of the cases C1 through C6 reflects a *policy* that the user is applying to resolve inconsistencies. We will discuss inconsistency management policies (IMPs for short) in detail in Chapter 3.

Second, we propose a unified framework for reasoning about inconsistency that extends the work in [SA07]. This framework applies to any monotonic logic, including ones for which inconsistency management has not been well studied (*e.g.*, temporal, spatial, and probabilistic logics), and the main goal is to allow end-users to bring their domain knowledge to bear by taking into account their preferences. In the example above neither

[1] This is more likely to happen, for example, when there is a scientific experiment with inconsistent data or when there is a critical action that must be taken, but cannot be taken on the basis of inconsistent data.

the bank manager nor the tax officer are making any attempt to find out the truth (thus far) about John's salary; however, both of them are making different decisions based on the same facts. The basic idea behind this framework is to construct what we call *options*, and then using a preference relation defined by the user to compute the set of *preferred options*, which are intended to support the conclusions to be drawn from the inconsistent knowledge base. Intuitively, an option is a set of formulas that is both consistent and closed with respect to consequence in a given monotonic logic. In [SA07] preferred options are consistent subsets of a knowledge base, whereas here this is not necessarily the case since a preferred option can be a consistent subset of the deductive closure of the knowledge base. We will present this framework in Chapter 4.

Applications dealing with the collection and analysis of news reports are highly affected by integration techniques, especially since millions of reports can be extracted daily by automatic means from different web sources. Oftentimes, even the same news source may provide widely varying data over a period of time about the same event. Past work on inconsistency management and paraconsistent logics assume that we have "clean" definitions of inconsistency. However, when reasoning about this type of data there is an extra layer of uncertainty that comes from the following two phenomena: (i) do two reports correspond to the same event or different ones?; and (ii) what does it mean for two event descriptions to be mutually inconsistent, given that these events are often described using linguistic terms that do not always have a uniquely accepted formal semantics? We propose a probabilistic logic programming language called PLINI (Probabilistic Logic for Inconsistent News Information) within which users can write rules specifying what they mean by inconsistency in situation (ii) above. Extensive work has also been done in duplicate record identification and elimination [BD83, HS95, ME97, CR02, BM03, BG04, BGMM$^+$09]. The main difference between our approach and previous work is the fact that the user is able to specify the notion of inconsistency that is of interest to him; furthermore, news reports are in general unstructured data containing complex linguistic modifiers which different users may interpret in different ways. We devote Chapter 5 to the treatment of this problem.

Another issue related to data integration is that of reasoning in the presence of incomplete information, or *null values*, in knowledge bases. Incomplete information can appear, just to give a common example, when merging knowledge bases with disparate schemas into a global schema. The consolidated knowledge base often contains null values for attributes that were not present in every source schema. Incomplete information makes the process of reasoning much harder since, if not treated carefully, results can present incorrect or biased information.

The problem of representing incomplete information in relational databases and understanding its meaning has been extensively studied since the beginnings of relational database theory Early work in this problem appears in [Cod74, Gra77, Gra79, Gra80, Lip79, Lip81]. Incomplete information is so widely spread in today's applications that practically any data analysis tool has to deal with null values in some way. Many data modeling and analysis techniques deal with missing values by removing from consideration whole records if one of the attribute values is missing, or using ad hoc methods of estimation for such values. Even though a wide variety of methods to deal with incomplete information have been proposed, which are in general highly tuned for particular applications, no tools have been provided to allow end-users to easily specify different ways of managing this type of data according to their needs and based on their expertise.

Consider another employee database and the following instance:

	Name	*Year*	*Department*	*Salary*	*Category*
t_1	$John$	2008	CS	$70K$	B
t_2	$John$	2009	CS	$80K$	B
t_3	$John$	2010	$Math$	?	A
t_4	$Mary$	2010	$Math$	$90K$	A

This relation contains a record for employee John for year 2009, in which the attribute *Salary* holds a null value, meaning that we do not know how much John earned that year. The classical approach in data cleaning and query answering would be to discard that record completely: since no information about the salary is provided, it is not possible to reason with that data. However, in many applications, users fill in this type of null values following strategies that are appropriate for the type of data, the type of application, or the decision process the application is supporting. For instance, a user of this database could decide to fill in the missing salary for John by using a regression model with the data for other years we have for the same employee and extrapolate a value for the missing year. Another user could decide to use the salary information from $Mary$, who was also in the *Math* department in 2010 and had the same category as John.

In this thesis, we propose a policy based framework to allow end-users to personalize the management of incomplete information by defining their own *Partial Information Policies* (PIPs for short) without having to depend on decisions made by, for instance, DBMS managers that might not know the data or the needs of the users. PIPs can be used in combination with relational operators allowing the user to issue queries that perform an assumption-based treatment of null values. This approach is developed in Chapter 6.

Finally, in the presence of structured and semi-structured knowledge bases such as relational databases, RDF databases, ontologies, etc., one important issue in the design of data integration systems is that of providing the users with a unified view of the different sources that they can query, making the whole process of integration transparent to the users. In such systems, the unified view is represented by a *target* or *mediated* schema. One of the main tasks in the design of such systems is to establish the mapping between the source schemas and the target schema. There has been intense work during the last few years on schema matching in order to answer queries over multiple databases [LC00, MKIS00, MHH$^+$01, DNH04, HMYW05, ESS05, Gal07, BMP$^+$08, CSD$^+$08, BV08]. More recently, there has been a realization that methods to automatically match schemas are uncertain. That is, when an algorithm for schema matching is executed, it might say that there are many possible mappings between one schema and another, and that a probability distribution over this set of mappings specifies the probability that a specific mapping is the correct one [DHY07]. Work to date [DHY07, DSDH08] on probabilistic schema mapping has studied the problem of answering SPJ (select-project-join) queries over multiple databases. In Chapter 7 we extend previous work by focusing on scalable algorithms for answering aggregate queries in the presence of probabilistic schema mappings. We also study how the use of uncertain schema mappings relates to inconsistency in the presence of integrity constraints.

In summary, the goal of this thesis is to attack particular problems of knowledge integration and provide personalizable approaches to handle them. Previous works in uncertainty and inconsistency management try to overcome these issues by considering that there is one correct way of handling them. Using the frameworks and tools proposed here, the users can specify when and how they want to manage/solve the issues that the integration of several heterogeneous knowledge bases yield, in the way that best suits their needs.

Chapter 2

Related Work

In this chapter, we will review the literature that is related to the work developed in this thesis. In Section 2.1, we will review the literature on inconsistency management both in artificial intelligence and database theory. Section 2.2 focuses on the work of partial information, particularly in relational databases. Section 2.3 will focus on the area of entity resolution and deduplication, which is related to our work described in Chapter 5 for identifying inconsistency in news reports about events. Finally, in support of the work presented in Chapter 7, Section 2.4 reviews the literature on the problem of data exchange and intregration, focusing on (probabilistic) schema matching.

2.1 Inconsistency Management

There has been a tremendous amount of work in inconsistency management since the 60s and 70s, when paraconsistent logics where introduced [dC74, BS89] and logics of inconsistency [Bel77, Gra78] were developed. In general, two kinds of approaches have been proposed in the literature for solving the problem of inconsistency.

The first type focuses on revising the knowledge base and restoring its consistency. This approach, initiated in [RM70], proposes to give up some formulas of the knowledge base in order to get one or several consistent subbases. More specifically, [RM70] considers expressing preferences among the maximal consistent subsets of the original (propositional) knowledge base, so that *preferred* maximal consistent subsets are determined. In the case of prioritized knowledge bases, Brewka [Bre89] has proposed a definition of a *preferred subbase*. The basic idea is to take as much important information into account as possible. More specifically, two generalizations of Poole's approach [Poo88] have been proposed: (1) stratified knowledge bases, in which formulas in the same stratum are equally preferred, whereas formulas in a stratum are preferred over formulas in lower strata; (2) the definition of a partial order among the formulas of a knowledge base.

Regarding inconsistency management based on a partial order on the formulas of a knowledge base, in [Roo92] the author defines the concept of a *reliability theory*, based on a partial *reliability relation* among the formulas in a first order logic knowledge base. In terms of this theory, the set of all *most reliable consistent sets of premisses* is defined. The set of theorems that can be proved from the theory is the set of propositions that are logically entailed by every most reliable consistent set. The work defines a special purpose logic based on first order calculus, and a deduction process to obtain the set of premises that can be *believed* to be true. The deduction process is based on the computation of justifications (premisses used in the derivation of contradictions) for believing or removing formulas, and it iteratively constructs and refines these justifications.

Priority-based management of inconsistent knowledge bases has been addressed in [BCD$^+$93] (see also [CLS95]). Specifically, propositional knowledge bases are considered and a knowledge base is supposed to be organized into strata, where each stratum has higher priority than the one directly below. Priorities in the knowledge base are used to select preferred consistent subbases. Inferences are made from the preferred subbases of the knowledge base, that is the knowledge base entails a formula ψ iff ψ can be classically inferred from every preferred subbase. In Section 4.6 we show how all of these works can be expressed in our general framework for inconsistency management, defined in Chapter 4.

The second approach to inconsistency accepts it and copes with it, prohibiting the logic from deriving trivial inferences; this includes multi-valued, paraconsistent, and default logics. The four valued logic in [Bel77] was used for handling inconsistency in logic programming [BS89] and was extended to the case of lattices [KS89, KS92] and bilattices [Fit91]. Subsequently, [Rei80a] introduced default logic — a database DB with integrity constraints IC could easily be written as a default logic theory $\Delta = (D, W)$ where $D = \{\frac{:f}{f} \mid f \in DB\}$ and $W = IC$ — the "extensions" of the default theory Δ correspond exactly to maximal consistent subsets of DB that are consistent with IC. Later, [BKM91, BKMS91] applied these ideas, together with algorithms, to integrate multiple knowledge bases. Kifer and Lozinskii [KL92] extended annotated logics of inconsistency developed by Blair and Subrahmanian [BS89] to handle a full first order case, while [TK93] developed similar extensions to handle inheritance networks.

Finally, argumentation methods [SL92, PS97, AP07, BH05, AC02] have been used for handling uncertainty and inconsistency by means of reasoning about how certain contradictory arguments defeat each other. Argumentation frameworks range from abstract argumentation to frameworks rooted in different types of logics. In abstract argumentation frameworks, proposed by Dung in [Dun95], arguments consist of just naming arguments along with an attack relation representing the fact that an argument is challenged by another, where no indication of the nature of that challenge is described in the relation. Approaches to logical argumentation include argumentation frameworks based on classical first-order argumentation [BH05], default logics [BDKT97], and defeasible logics [GS04, PS97], among others.

The area of belief change in artificial intelligence is closely related to the management of inconsistent information. *Belief change* aims to adequately model changes in knowledge in belief systems, *i.e.*, modeling the dynamics of the knowledge that constitutes the set of beliefs of an agent when new information is presented. There are different kinds of belief changes (belief change operations): revision, contraction, and consolidation.

It seems reasonable to think that inconsistency management techniques are materializations of certain variations of belief change methods. However, the seminal works on belief change [AGM85] are done under the assumption that the knowledge base to be contracted or revised is a *belief set*, *i.e.*, a set of sentences that is deductively closed. In the setting of a relational database, considering the logical closure of the first-order encoding of a database instance as the belief set to be revised or consolidated is not really practical since we would be considering as part of the knowledge base elements that do not really have independent standing in the relational instance. For instance, for every record of the form $salary(John, 80K)$, the sentence $salary(John, 80K) \vee \phi$, where ϕ is any other sentence that can be formed from the language, will hold. The latter sentence is a mere logical consequence that should have no standing of its own, especially in relational databases where we only desire to reason about the literals explicitly present as records in the database. Knowledge bases in this setting are therefore *belief bases*, *i.e.*, sets of sentences that are not closed under logical consequence.

The AGM model proposed in [AGM85] presents a set of postulates for contraction and revision operations. In [Gar78], the basic set of postulates for contraction and revision operations were first presented; later, in [AGM85] , the set of postulates were extended and *partial meet contraction* over belief sets was presented together with a representation theorem that shows that partial meet contraction exactly characterizes the set of contraction operators that satisfy the rationality postulates from [Gar78]. Unfortunately, not all rationality postulates and belief change operators that satisfy the postulates from the AGM model [AGM85, Gar88b], and its derivations, are applicable to belief bases. Alternatively, Hansson [Han94, Han97] defines a different set of rationality postulates, and corresponding partial meet revision and contraction operators, to deal exclusively with belief bases. In Section 3.5, we discuss the relationship between belief revision in belief bases and IMPs, by studying the rationality postulates from [Han97] if we consider an IMP as a belief revision operator. Partial meet contraction and revision as defined above is actually applicable for both belief sets and belief bases. However, it is important to note that $K \perp \alpha$ is the set of maximal subsets of K that do not imply α, and it is not enough that they do not contain α. Consider as an example the belief base $K = \{p \vee q, p \leftrightarrow q\}$, in this case $K \perp p = \{\{p \vee q\}, \{p \leftrightarrow q\}\}$. Repairing a relational database by keeping the intersection of all maximal consistent subsets w.r.t. a set of integrity constraints can be shown to be a special case of partial meet revision (*i.e.*, full meet revision).

Finally, there are also several important works [Loz94, HK05, GH06, MPS$^+$07] on measuring the amount of inconsistency in a database or knowledge base that can be used in combination with any of the inconsistency management techniques described above.

2.1.1 Database Repairing and Consistent Query Answering

In databases, the integrity constraints (ICs) capture the semantics of data with respect to the external reality that the database is expected to model. Therefore, an inconsistent database instance, together with the integrity constraints, may be represented as an inconsistent set of formulas, where integrity constraints are closed first-order formulas. Examples of the most common types of ICs treated in the literature are: (1) *Universal integrity constraints*: allow the expression of constraints such as "the salary of every employee must be greater than or equal to zero"; (2) *Denial constraints*: allow the expression of constraints such as "it is not possible for an employee to be married and single at the same time"; (3) *Functional Dependencies*: allow expressions like "if two books have the same ISBN then their titles must be the same"; and (4) *Inclusion dependencies*: allow expressions like "if a person is registered as a graduate student in the *GradStudents* relation, then there must exist a record for that same person in the *Students* relation".

A database instance I is said to be consistent w.r.t. a set of integrity constraints if it satisfies the given set IC (in the standard model-theoretic sense), and inconsistent otherwise. Methods to clean data and/or provide consistent query answers in the presence of inconsistent data have been widely studied in the last decades [FUV83, FKUV86, JDR99, SS03, Cho07, BFFR05]. [FUV83, FKUV86] introduced the use of maximal consistent subsets (called "flocks") to accommodate updates to a database that violate integrity constraints.

The field of *database repairing* and *consistent query answering* (CQA for short) has gained much attention since the work of Arenas et al. [ABC99], which provided a model-theoretic construct of a database *repair*: a repair of an inconsistent database is a model of the set of ICs that is minimal, *i.e.*, "as close as possible" to the original database. Clearly, repairs may not be unique, and in the general case there can be a very large number of them. The most widely accepted semantics for querying a possible inconsistent database is that of *consistent answers*. A consistent answer for a query over a possibly inconsistent database is comprised of the set of tuples that appear in the answer to the query over *every* possible repair. CQA enforces consistency at query time as an alternative to enforcing it at the instance level as conventional data cleaning techniques do. This allows us to focus on a smaller portion of the database for which repairs can be computed more easily.

Furthermore, techniques have been developed so it is not necessary to materialize every possible repair.

The work of [Cho07] addresses the basic concepts and results of the area of consistent query answering. They consider universal and binary integrity constraints, denial constraints, functional dependencies, and referential integrity constraints (*e.g.*, foreign key constraints). [BFFR05] presents a cost-based framework that allows finding "good" repairs for databases that exhibit inconsistencies in the form of violations to either functional or inclusion dependencies. They propose heuristic approaches to constructing repairs based on equivalence classes of attribute values; the algorithms presented are based on greedy selection of least repair cost, and a number of performance optimizations are also explored. The technique based on query rewriting introduced in [ABC99] for quantifier-free conjunctive queries and binary universal constraints was extended in [FM05, FFM05] to work for a subclass of conjunctive queries in the presence of key constraints. The complexity of the consistent query answer problem was investigated in [CLR03] in the presence of both functional and inclusion dependencies, and further studied in [CM05] in the presence of denial constraints and inclusion dependencies. The notion of consistent answer was extended to the case of aggregate queries in [ABC$^+$03b], where the evaluation of consistent answers of aggregate queries was investigated in the presence of functional dependencies. The logic-based frameworks in [ABC03a, GGZ03, BB03] assume that tuple insertions and deletions are the basic primitives for repairing inconsistent data. Repairs also consisting of value-update operations were considered in [FLPL$^+$01, BBFL05, Wij03, Wij05, BFFR05, FFP05, FFP07]. In particular, [Wij03, Wij05, FFP05] investigated the complexity of the consistent query answering problem when the basic primitive for repairing data is the attribute-value update, whereas [FLPL$^+$01, BBFL05, BFFR05, FFP07] focused on the problem of computing repairs rather than computing consistent answers.

In CQA, an answer to a given query posed to an inconsistent database is said to be consistent if the same answer is obtained from every possible repair of the database. Clearly, this is a very cautious approach, and it can yield a great loss of information. Even though several works investigated the problem of repairing and querying inconsistent data considering different classes of queries and constraints, only recently there have been two proposals which shifted attention towards improving the quality of consistent answers [CGZ09, SCM09]. These approaches developed more specific repairing strategies that reduce the number of possible repairs to be considered and improve their quality according to some criteria specified by the database administrator on the basis of users' preferences. We study these two approaches more in depth in Section 3.6 in comparison with IMPs, our approach to inconsistency management in relational databases developed in Section 3.5.

Almost all past approaches described above proceeded under the assumption that there was some "epistemically correct" way of resolving inconsistencies or reasoning in the presence of inconsistency. More recently, [SA07] argued that inconsistency can often be resolved in different ways based on what the user wants, and they provided a mechanism to reason about maximal consistent subsets using objective functions that the user gets to choose.

With the exception of [SA07], to the best of our knowledge all past work on inconsistency management in databases assumes that the database designer knows more than the user about how to handle inconsistency, and hence, inconsistencies are resolved within the database infrastructure. Unfortunately, this is not wise. A database designer for Oracle is not likely to know how, for example, an astronomy database was collected, what assumptions were made when collecting it, and how inconsistencies should be handled in the context of such knowledge. In the same vein, a database developer for other major DB vendors may not understand how some epidemiological data in a government's Health Ministry was collected, and what the ramifications and risks are to a policy-maker using that data to make decisions. The policy-maker may wish to make decisions on the basis of some inconsistent data taking into account not only information about how the data was collected, which sources were reliable, and so forth, but also about his own risk if he makes the wrong decision. A database designer who has embedded some form of inconsistency management within the DB infrastructure will not know this a priori. This is the main difference between the approaches described in this section and our proposals of IMPs, in Chapter 3, and the general framework for inconsistency management in Chapter 4.

2.2 Partial Information

Incomplete information can appear in knowledge bases for very different reasons. For instance, people filling out forms or surveys often leave fields incomplete, in many cases due to security or privacy issues. Data integration techniques are also other sources of incomplete information, especially when performed automatically. For instance, when merging knowledge bases with disparate schemas into a global schema, the consolidated knowledge base often contains null values for attributes that were not present in every source schema. The problem of representing incomplete information in relational databases has been extensively studied since the introduction of the relational data model [Cod74, Bis79, Cod79, Gra77, IJ81, IL84b, AM86, LL99, Lie82, Jr.79, Zan84]. Incomplete information makes the process of reasoning much harder since, if not treated carefully, results can present incorrect or biased information.

It was recognized early on that there are different types of null values, each of which reflects different intuitions about why a particular piece of information is missing. Different types of null values have been proposed in the literature. [CGS97a] presents the following list of types of null values:

- *Existential Null.* The value for an attribute A exists but it is not known. The actual value for an existential null could be any value within the domain of A.
- *Maybe Null.* The value for an attribute A may exist but it is not known at the moment of the record's creation or may not exist at all, and it is not known which is actually the case.
- *Place holder Null.* An attribute might not be applicable for a certain record.
- *Partial Nulls.* There exists a value for an attribute but it might be one out of a set of possible values. The set of possible values is a subset of the attribute's domain.
- *Partial Maybe Nulls.* An attribute may or may not be applicable for a particular record, but if it is applicable then there exists a value and it must fall within a specified set.

There is a clear trade-off between the expressivity of the model for incomplete information and the difficulty of answering queries. Allowing more than one kind of null value makes reasoning over the data even more complex, especially when several relations are put together, since there might be inconsistency at the null value level. For instance, in one relation the value for attribute $Phone$ for an employee appears as an existential null whereas in another database the record for the same employee contains a null value for $Phone$ but as a place holder. [CGS97a] presents a unified framework to deal with different kinds of null values.

No-information nulls were introduced in [Zan84] to deal with the case where it is not known whether a missing value exists or not. They have also been considered in [Lie82] and [AM86, HL10], where integrity constraints in the presence of no-information nulls are studied.

A greater amount of work has been devoted to the study of databases containing only unknown nulls. In this context, different problems have been addressed, such as query answering [AKG91, Gra91, IL84a, Rei86, YC88], the characterization of consistency in the presence of integrity constraints [Gra91, IL83, LL98, LL99, Vas80] and updating the database [AG85, Gra91]. As stated in [IL84a], a condition for correctness is that if a null value has a specified semantic interpretation, that is, if we assume that a specified relation exists between a table with nulls and the real world, then this relation should be similar for the tables that are arguments of a relational operator and for the relation obtained as the result (this is the requirement for a *strong representation system*).

Codd's approach [Cod74] is based on a three-valued logic with truth values *True*, *False*, and *Unknown*. Under this approach, a condition over a relation containing null values can evaluate to *Unknown* depending on the logical operators. For instance, $Unknown \vee Unknown$ evaluates to *Unknown*. Codd's semantics has been criticized on semantic grounds by Grant [Gra77] and Lipski [Lip79]. Other works such as [Lip79, Lip81, Vas79] propose different semantics for other subsets of the relational operators. Through the definition of a representation system, [IL84a] formally presents the conditions that must be satisfied in any semantically meaningful extension of the relational algebra to handle incomplete information.

More expressive data models where the values that (labeled) unknown nulls can take can be constrained have been considered in [AKG91, Gra91, IL84b]. Codd tables, V-tables, and C-tables are analyzed in these works as possible representation systems. In Codd tables, null values are represented by a special symbol, and the semantics for that symbol is that it is an *unknown* value. Codd tables are a strong representation system for projection and selection. V-tables allows many different ("marked") null values, or variables. The meaning of a null value X is that the value is unknown but it is the same value every time X appears. For this representation system, [IL84a] shows that V-tables are a strong representation system for projection, positive selection, union, join, and renaming of attributes; therefore, arbitrary conjunctive queries can be processed correctly. Finally, C-tables (or conditional tables) are V-tables with an additional column, *condition*, containing a condition for the variables that represent null values. C-tables are a strong representation system for projection, selection, union, join, and renaming.

The most common adopted semantics for this kind of databases is given in terms of completions, or *possible worlds* (we will provide the formal definition in Chapter 6). Two largely accepted semantics of query answering are: (i) *certain answers* – a tuple is a certain answer to a query Q if it is an answer to Q in every completion; (ii) *possible answers* – a tuple is a possible answer to a query Q if it is an answer to Q in some completion.

The main difference between the aforementioned approaches and the treatment of partial information that we propose in Chapter 6, is that in the former no attempt at resolving incompleteness is made. On the contrary, in Chapter 6, we argue that data stored in the database and the knowledge the user has of it can be profitably exploited to add valuable new information to the database.

Outside the database theory community, the fields of data mining, data warehousing, and data management have addressed the problem of data quality in the presence of incomplete information mainly providing approaches based on cleaning techniques. In general, simple estimates for numeric and categorical data are used [MST94, Pyl99, Qui93]. A more complex approach in classification analysis was proposed in [BFOS84]. For numeric data analysis, methods based on regression techniques were also developed [FLMC02, Pyl99]. Probabilistic methods include probabilistic weighting methods such as the one proposed in [Qui93] and Bayesian methods. In the last category, [CS86] proposes a Bayesian procedure for estimating and replacing missing data based on some prior knowledge about the distribution of the data; [CA03] estimates and replaces null values for categorical attributes using the uniform prior distribution and a Dirichlet posterior distribution. Recently, in [Li09] the posterior probabilities of a missing value belonging to a certain category are estimated using the simple Bayesian method and are used to compute a replacement value.

The main difference between the works above and our approach is that the former are ad-hoc statistical approaches tailored for particular domains and applications; in contrast, we develop a more general *formal* framework that allows users to specify a wide set of strategies, and in particular those that he believes are adequate for the specific data and application at hand. Furthermore, all of these works are designed, in general, to be applied as “cleaning tools” to a database and they do not study the interaction of the methods with relational algebra operators as we do in this work. Another difference is that we adopt a richer data model which allows us to express different kinds of partial information, whereas all the approaches mentioned above deal with unknown nulls only. Our work also addresses the problem of supporting the efficient application of incompleteness resolution methods, by providing index structures that allow us to manage large databases, which is an issue addressed in none of the works above.

Finally, we mention that our work in Chapter 6 is also related to hypothetical reasoning [Res64], where one considers what follows from an assumption. Usually the assumption is inconsistent with known information, requiring changes to the latter trying to make it consistent with the assumption. In our case we may consider filling in a value as a hypothesis.

Finally, many works in AI have tackled the issue of dealing with incompleteness. These include logic program completions [Cla77], closed world assumption [Rei78], auto-epistemic logic [Moo85], default logic [Rei80b], alternating-time temporal logic with imperfect information [JD05, JD06], to mention a few. In these works, partial information is defined in a different way, in a kind of uncertain information that does not deal with null values.

2.3 Entity Resolution and Deduplication

The problem of event equivalence identification addressed in Chapter 5 is closely related to a class of problems called *entity resolution* in machine learning [BG07]. Given a set of (potentially different types of) entities, entity resolution asks for a partition of this set such that entities are grouped together iff they are equivalent. In our problem, the entities of primary interest are events, but we also reason about the equivalence of actors, locations, and objects, which are entities of secondary interest. Traditional machine learning approaches to entity resolution focus on pairwise entity equivalence determination using established techniques such as Bayesian networks [Hec98], support vector machines [CS00], or logistic regression [NJ02]. Hence, for each pair of entities a classifier is used to determine equivalence. In a post processing step, inconsistencies due to violations of transitivity are resolved.

Recent work has considered joint entity resolution in the case when entities are related [GD05]. Such approaches are termed *relational*, because they determine all event equivalences at once and take relationships between entities into account instead of making a series of independent decisions. Some proposals for relational entity resolution define a joint probability distribution over the space of entity equivalences and approximate the most likely configuration using sampling techniques, such as Gibbs sampling [SNB$^+$08], or message passing algorithms, such as loopy belief propagation [MWJ99]. While these approaches are based on a probabilistic model, they provide no convergence guarantee and allow little theoretical analysis. Other relational approaches are purely procedural in that they apply non-relational classifiers in an iterative fashion until some convergence criterion is met. While iterative methods are fast in practice, they are not amenable to theoretical analysis or convergence guarantees. All these approaches are feature driven and do not have a formal semantics.

Recently, [BGMM$^+$09] proposed a generic approach to entity resolution (ER). In this work the authors formalize the generic ER problem, treating the functions for comparing and merging records as black-boxes. They identify four important properties that, if satisfied by the match and merge functions, enable much more efficient ER algorithms: idempotence, commutativity, associativity, and representativity (a record obtained from merging two records represents the original records, in the sense that any other record that would have matched the first two will also match it). Three efficient ER algorithms are developed: G-Swoosh for the case where the four properties do not hold, and R-Swoosh and F-Swoosh that exploit the 4 properties. F-Swoosh in addition assumes knowledge of the "features" (e.g., attributes) used by the match function. R-Swoosh (and F-Swoosh) can be used also when the four match and merge properties do not hold, if an approximate result is acceptable.

2.4 Schema Mappings and Data Exchange

Data exchange is the problem of taking data structured under a source schema and translating into an instance of a target schema that reflects the source data as accurately as possible. Data exchange is an old and recurrent problem in databases. The work of [FKMP05] reviews the principal components and solutions to the problem, focusing on query answering. In data exchange, it is assumed that there is a set of constraints that define the relationship between the source schema and the target schema. These constraints, called *source-to-target* dependencies, establish how and what source data should appear in the target. The data exchange problem can then be specified as: giving an instance I over a source schema S, materialize an instance J over schema target T such that J satisfies the set of source-to-target dependencies between S and T. The problem of query answering in a data exchange setting has been also studied in the last decade. [FKP05, GN06] focus on complexity results for computing (universal) certain answers, in the presence of *tuple generating* and *equality generating* dependencies, apart from a set of source-to-target dependencies.

A related problem is that of *data integration*, where the goal is to query heterogeneous data in different sources via a virtual *global* schema. There has been intense work during the last few years on schema matching in order to answer queries over multiple databases [LC00, MKIS00, MHH$^+$01, DNH04, HMYW05, ESS05, Gal07, BMP$^+$08, CSD$^+$08, BV08]. The works mentioned above assume that it is possible to exactly map the translation. However, the work of [DHY07] argues that this is not necessarily always the case and that therefore uncertainty on how to map the source schema to the target schema may arise. For instance, methods to automatically match schemas are uncertain — when an algorithm for schema matching is executed, it might determine that there are many possible mappings between one schema and another, and that a probability distribution over this set of mappings specifies the probability that a specific mapping is the correct one [DHY07, Gal06]. For example, in a residential or commercial real estate web site that aggregates information from multiple realtors across the country, there is a need to find mappings between disparate schemas. In such a case, there may be many different ways of mapping one schema to another and a probability distribution might tell us the probability that a given mapping is correct. Alternatively, a web search engine doing a product search over the databases of multiple vendors needs to find mappings between the product database of one vendor and the product database of another vendor. Multiple ways of representing the data might lead to multiple possible schema mappings, together with a probability distribution over the set of mappings.

Work to date [DHY07, DSDH08] on probabilistic schema mapping has studied two semantics for answering queries over multiple databases using probabilistic schema matches — a "by-table" semantics and a "by-tuple" semantics. In the "by-table" semantics a single mapping should be applied to the entire set of tuples in the source relation. Each query is answered separately and the answers are assigned a probability according to the mappings that produced it. However, in the "by-tuple" semantics a choice of mapping must be made for each tuple. So it is necessary to consider all possible ways of assigning a mapping to a tuple, and answer the query for each one of them. Clearly, by-tuple semantics is more complex and in the general case difficult to compute.

When aggregate queries are considered, then in addition to whether a by-table or by-tuple semantics should be used, we need to consider the semantics of the aggregates themselves in the presence of uncertainty. Some work has been done on aggregates over uncertain data [RSG05, JKV07], yet none at all w.r.t. aggregate computations under probabilistic schema mapping. In Chapter 7 we study answering aggregate queries in this setting.

Chapter 3

Inconsistency Management Policies

The work described in this chapter appears in [MPP$^+$08, MPP$^+$10].

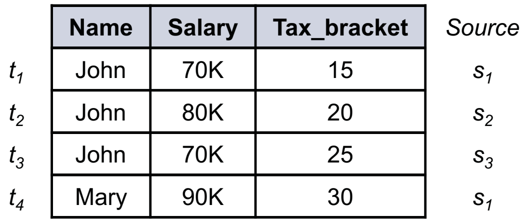
3.1 Introduction and Motivating Example

Almost all past approaches to inconsistency proceeded under the assumption that there was some "epistemically correct" way of resolving inconsistencies or reasoning in the presence of inconsistency. More recently, [SA07] argued that inconsistency can often be resolved in different ways based on what the user wants, and they provided a mechanism to reason about maximal consistent subsets (also called "repairs" by others such as [ABC99]) using objective functions that the user gets to choose.

With the exception of [GH92, GH93], and [SA07], *all* past work on inconsistency management in databases assumes that the database designer knows more than the user about how to handle inconsistency, and hence, inconsistencies are resolved within the database infrastructure. Unfortunately, this is not wise. A database designer for Oracle is not likely to know how, for example, an astronomy database was collected, what assumptions were made when collecting it, and how inconsistencies should be handled in the context of such knowledge. In the same vein, a database developer for other major DB vendors may not understand how some epidemiological data in a government's Health Ministry was collected, and what the ramifications and risks are to a policy maker using that data to make decisions. The policy-maker may wish to make decisions on the basis of some inconsistent data taking into account not only information about how the data was collected, which sources were reliable, and so forth, but also about his own risk if he makes the wrong decision. A database designer who has embedded some form of inconsistency management within the DB infrastructure will not know this a priori.

The work in this chapter aims to support the two types of users mentioned above. The main goal of this work is developing the theory and implementation support required in databases so that *end users can bring both their application specific knowledge as well as their own personal risk to bear when resolving inconsistencies*. To reiterate why this is important, let us consider the following example.

Example 1. *The Emp relation below represents data about employees, their salaries and tax brackets; each tuple t_i in the relation is provided by a source s_j (which is annotated on the same row).*



	Name	Salary	Tax_bracket	*Source*
t_1	John	70K	15	s_1
t_2	John	80K	20	s_2
t_3	John	70K	25	s_3
t_4	Mary	90K	30	s_1

Let us assume that salaries are uniquely determined by names. In this case, a user may want to resolve the inconsistency about John's salary in many different ways.

C1 If he were considering John for a loan, he might want to choose the lowest possible salary of John to base his loan on. Here, the user's decision is based on the *risk* he would take if he gave John a loan amount based on a higher income.

C2 If he were assessing the amount of taxes John has to pay, he may choose the highest possible salary John may have — in this case, the end user is basing his decision on his own *reward* (assuming his salary is somehow based on the amount of additional taxes he can collect).

C3 If he were just trying to estimate John's salary, he may choose some number between 70K and 80K (*e.g.*, the average of the three reports of John's salary) as the number.

C4 If he has different degrees of confidence in the sources that provided these salaries, he might choose a weighted mean of these salaries.

C5 He might choose not to resolve the inconsistency at all, but to just let it persist until he can clear it up – there may be no need to deal with John's salary with respect to his current task.

C6 He might simply consider *all* the data about John unreliable and might want to ignore it until it can be cleared up – this is the philosophy of throwing away all contaminated data.

Each of cases C1 through C6 reflects a *policy* that the user is using to resolve inconsistencies. We are not aware of a single piece of past work that can handle all six reasonable possibilities mentioned above. [BKM91, SA07, ABC99, BFFR05, CLR03, BC03, Cho07] can handle cases C1 and C2, but not the other cases. Much as a carpenter has tools like hammers and saws, we propose to put data cleaning policies in the hands of users so that they can use them as tools, when appropriate, for reasoning about the data. *It is important to enable users to bring their application specific knowledge to bear when resolving inconsistencies*. Section 2.1 describes work in AI and databases that relate to the work developed in this chapter.

The contributions of this chapter are as follows: in Section 3.3 we first define the concept of a *policy* for managing inconsistency. Our notion of an inconsistency management policy generalizes [BKM91, BKMS91, ABC99] by allowing policies that either remove inconsistency completely or allow part or all of the inconsistency to persist. Our notion of a policy accounts for all six cases above, and many more. We start with policies applicable to a single functional dependency (FD for short), one of the most common kinds of integrity constraints used in databases, then we discuss policies to manage multiple FDs. In Sections 3.5 and 3.6 we give a detailed overview and results of how our framework relates to postulates for the revision of *belief bases* [Han97] and to recent research in the area of *consistent query answering* [CGZ09, SCM09]. In Section 3.7 we show that our policies can be embedded as operators within the relational algebra in two different ways – one where the policy is applied first (before other relational operators) and another where it is applied last. We study the interaction of these policy usage methods together with the standard relational operators and provide several interesting results. In Section 3.8 we present several approaches to efficiently implement an IMP-based framework. Finally, Section 3.9 outlines conclusions.

3.2 Syntax and Notation

We assume the existence of relational schemas of the form $S(A_1, \dots, A_n)$ [Ull88] where the A_i's are attributes. Each attribute A_i has an associated domain, $dom(A_i)$. A *tuple* over S is a member of $dom(A_1) \times \cdots \times dom(A_n)$, and a set of such tuples is called a *relation*. We use $t[A_i]$ to denote the value of the A_i attribute of tuple t. We use *Attr*(S) to denote the set of all attributes in S.

Given the relational schema $S(A_1, \ldots, A_n)$, a functional dependency (FD) *fd* over S is an expression of the form $A'_1, \ldots, A'_k \to A'_{k+1}, \ldots, A'_m$, where $\{A'_1, \ldots, A'_m\} \subseteq$ *Attr*(S). A relation R over the schema S *satisfies* the above FD iff $\forall\, t_1, t_2 \in R$, $t_1[A'_1] = t_2[A'_1] \wedge \ldots \wedge t_1[A'_k] = t_2[A'_k] \Rightarrow t_1[A'_{k+1}] = t_2[A'_{k+1}] \wedge \ldots \wedge t_1[A'_m] = t_2[A'_m]$. Without loss of generality, we assume that every functional dependency fd has exactly one attribute on the right-hand side (i.e., $k+1=m$) and denote this attribute as $RHS(fd)$. Moreover, with a little abuse of notation, we write that fd is defined over R.

Definition 1. *Let R be a relation and $\mathcal{F}$ a set of functional dependencies. A* culprit *is a set $c \subseteq R$ not satisfying $\mathcal{F}$ such that $\forall\, c' \subset c$, c' satisfies $\mathcal{F}$.*

For instance, the culprits in the example of the Introduction are $\{t_1, t_2\}$ and $\{t_2, t_3\}$. We use $culprits(R, \mathcal{F})$ to denote the set of culprits in R w.r.t. $\mathcal{F}$.

Definition 2. *Let R be a relation and $\mathcal{F}$ a set of functional dependencies. Given two culprits $c, c' \in culprits(R, \mathcal{F})$, we say that c and c'* overlap*, denoted $c \bigtriangleup c'$, iff $c \cap c' \neq \emptyset$.*

Definition 3. *Let $\bigtriangleup^*$ be the reflexive transitive closure of relation $\bigtriangleup$. A* cluster *is a set $cl = \bigcup_{c \in e} c$ where e is an equivalence class of $\bigtriangleup^*$.*

In the example of the Introduction, the only cluster is $\{t_1, t_2, t_3\}$. We will denote the set of all clusters in R w.r.t. $\mathcal{F}$ as *clusters*$(R, \mathcal{F})$.

3.3 Inconsistency Management Policies

In this section, we introduce the concept of *policy* for managing inconsistency in databases violating a given set of functional dependencies. Basically, applying an inconsistency management policy on a relation results in a new relation with the intention of obtaining a lower degree of inconsistency.

Definition 4. *An inconsistency management policy (IMP for short) for a relation R w.r.t. a set $\mathcal{F}$ of functional dependencies over R is a function $\gamma_{\mathcal{F}}$ from R to a relation $R' = \gamma_{\mathcal{F}}(R)$ that satisfies the following axioms:*

Axiom A1 *If $t \in R - \bigcup_{c \in culprits(R,\mathcal{F})} c$, then $t \in R'$. This axiom says that tuples that do not belong to any culprit cannot be eliminated or changed.*

Axiom A2 *If $t' \in R' - R$, then there exists a cluster cl and a tuple $t \in cl$ such that for each attribute A not appearing in any fd $\in \mathcal{F}$, $t.A = t'.A$. This axiom says that every tuple in R' must somehow be linked to a tuple in R.*

Axiom A3 *$\forall fd \in \mathcal{F}$, $|culprits(R, \{fd\})| \geq |culprits(R', \{fd\})|$. This axiom says that the IMP cannot increase the number of culprits.*

Axiom A4 *$|R| \geq |R'|$. This axiom says that the IMP cannot increase the cardinality of the relation.*

$\gamma_{\mathcal{F}}$ is a *singular* IMP iff $\mathcal{F}$ is a singleton. When $\mathcal{F} = \{fd\}$ we write γ_{fd} instead of $\gamma_{\{fd\}}$.

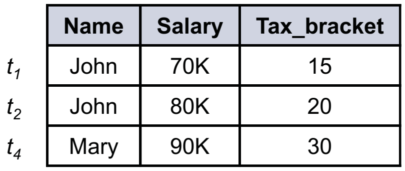
It is important to note that Axioms A1 through A4 above are not meant to be exhaustive. They represent a minimal set of conditions that we believe any inconsistency management policy should satisfy. Specific policies may satisfy additional properties.

3.3.1 Singular IMPs

In this section, we define three important *families* of singular IMPs (*tuple-based*, *value-based*, and *interval-based*), which satisfy Axioms A1 through A4 and cover many possible real world scenarios. Clearly, Definition 4 allows to specify many other kinds of IMPs, based on the user's needs.

Definition 5 (tuple-based family of policies)**.** *An IMP τ_{fd} for a relation R w.r.t. a functional dependency fd is said to be a* tuple-based policy *if each cluster $cl \in clusters(R, \{fd\})$ is replaced by $cl' \subseteq cl$ in $\tau_{fd}(R)$.*

Tuple-based IMPs generalize the well known notion of maximal consistent subsets [BKM91] and repairs [ABC99] by allowing a cluster to be replaced by any subset of the same cluster. Notice that tuple-based IMPs allow inconsistency to persist – a user may choose to retain all inconsistency (case C5) or retain part of the inconsistency. For instance, if the user believes only sources s_1, s_2 in Example 1, he might choose to replace the cluster $\{t_1, t_2, t_3\}$ by the cluster $\{t_1, t_2\}$ as shown below.



	Name	Salary	Tax_bracket
t_1	John	70K	15
t_2	John	80K	20
t_4	Mary	90K	30

[BKM91, ABC99] do not allow this possibility. Observe that this kind of policy can cause some information to be lost as a side effect. In our example, although the *Tax_bracket* 25 is not involved in any FD, it is lost when the policy is applied. We now introduce two kinds of policies that avoid this problem. The first kind of policy is based on the notion of *cluster simplification*.

Definition 6. *Given a cluster $cl \in clusters(R, \{fd\})$, cl' is a* cluster simplification *of cl iff $\forall\, t_1, t_2 \in cl$ such that $t_1[RHS(fd)] = t_2[RHS(fd)]$, either $t_1, t_2 \in cl'$ or there exist $t_1', t_2' \in cl'$ obtained from tuples t_1, t_2 by replacing $t_1[RHS(fd)]$ and $t_2[RHS(fd)]$ with $t_3[RHS(fd)]$ where $t_3 \in cl$.*

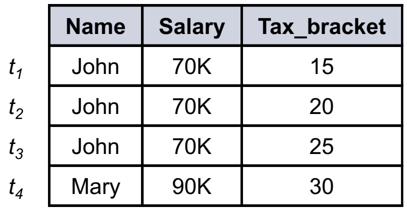
A simplification allows replacement of values in tuples in the same cluster (in the attribute associated with the right-hand side of an FD).

Example 2. *A cluster simplification of the cluster $cl = \{t_1, t_2, t_3\}$ of Example 1 may be the cluster $cl' = \{t'_1, t_2, t'_3\}$ where t'_1 and t'_3 are obtained from t_1 and t_3 by replacing $t'_1[Salary] = t'_3[Salary] = 70K$ with the value $t_2[Salary] = 80K$.*

This leads to the following kind of IMP.

Definition 7 (value-based family of policies)**.** *An IMP ν_{fd} for a relation R w.r.t. a functional dependency fd is said to be a* value-based policy *if each cluster $cl \in clusters(R, \{fd\})$ is replaced by a cluster simplification of cl in $\nu_{fd}(R)$.*

Thus, a value-based IMP either leaves a cluster unchanged or reduces the number of distinct values for the attribute in the right-hand side of the functional dependency. A user may, for example, decide to use his knowledge that s_1 reflects more recent information than s_2 to reset the s_2 information to that provided by s_1. In this case, the relation returned by the value-based policy is:



	Name	Salary	Tax_bracket
t_1	John	70K	15
t_2	John	70K	20
t_3	John	70K	25
t_4	Mary	90K	30

We now show that value-based policies satisfy Axiom A3, by deriving the number of culprits in a cluster.

Theorem 1. *Let R be a relation over the relational schema $S(A_1, \ldots, A_n)$ and $fd : A'_1, \ldots, A'_k \rightarrow A'_{k+1}$ with $\{A'_1, \ldots, A'_{k+1}\} \subseteq Attr(S)$ a FD over S. For each $cl \in clusters(R, \{fd\})$, assume that the values $t[A'_{k+1}]$ of tuples $t \in cl$ are the union of single-value multi-sets $V_1, V_2, \ldots, V_\ell$ (where every multi-set V_i contains the single value v_i with cardinality C_i). Then:*

1. $|culprits(cl, \{fd\})| = \sum_{i<j} C_i C_j$;

2. *$|culprits(cl', \{fd\})| \leq |culprits(cl, \{fd\})|$, where cl' is a cluster simplification of cl.*

Proof. The results follow from the fact that a cluster can be viewed as a complete ℓ-partite graph having vertices corresponding to values in $V_1, V_2, \ldots, V_\ell$ where each edge represents a culprit. The number of edges in this complete ℓ-partite graph is the number of possible edges in the complete graph decreased by the sum of edges that could be in every multi-set V_i:

$$|\mathit{culprits}(cl, \{\mathit{fd}\})| = \frac{(\sum_i C_i)((\sum_i C_i) - 1)}{2} - \sum_i \frac{C_i(C_i - 1)}{2}$$

$$= \frac{(C_1 + C_2 + \cdots + C_k)(C_1 + C_2 + \cdots + C_k - 1)}{2} - \sum_i \frac{C_i^2 - C_i}{2}$$

$$= \frac{\sum_i C_i^2 + \sum_{i \neq j} C_i C_j - \sum_i C_i}{2} - \frac{\sum_i {C_i}^2}{2} + \frac{\sum_i C_i}{2} = \frac{\sum_{i \neq j} C_i C_j}{2} = \sum_{i<j} C_i C_j$$

As $\left(\sum_i C_i\right)^2 = \sum_i {C_i}^2 + \sum_{i<j} 2C_i C_j$ we obtain

$$|\mathit{culprits}(cl, \{\mathit{fd}\})| = \sum_{i<j} C_i C_j = \frac{(\sum_i C_i)^2 - \sum_i {C_i}^2}{2}.$$

With reference to Definition 6, it is easy to see that (*i*) the sum of cardinality C_i of the multisets V_i does not change after a cluster simplification, that is, $\sum_i C_i$ does not change, and therefore, $(\sum_i C_i)^2$ is constant; (*ii*) every time there is a substitution of values v_a on $RHS(\mathit{fd})$ of a group of tuples with values v_b on $RHS(\mathit{fd})$ of another group of tuples, the two multisets V_a, V_b collapse into a single multiset whose cardinality is $C_a + C_b$. Hence, after a cluster simplification, it is the case that

$$|\mathit{culprits}(cl', \{\mathit{fd}\})| = \frac{(\sum_i C_i)^2 - \sum_{i \neq a, i \neq b} {C_i}^2 - (C_a + C_b)^2}{2}$$

$$= \frac{(\sum_i C_i)^2 - \sum_{i \neq a, i \neq b} {C_i}^2 - {C_a}^2 - {C_b}^2 - 2C_aC_b}{2} = \frac{(\sum_i C_i)^2 - \sum_i {C_i}^2 - 2C_aC_b}{2}.$$

Therefore, after a cluster simplification which substitutes the values in V_a with those in V_b, the number of culprits decreases by C_aC_b. □

The third family of policies we present are *interval-based* policies.

Definition 8 (interval-based family of policies)**.** *An IMP ξ_{fd} for a relation R w.r.t. a functional dependency fd is said to be an* interval-based policy *if $\forall cl \in clusters(R, \{fd\})$, cl is replaced by a set cl' such that either $cl' = cl$ or $cl' = cl \setminus \{t_1, \dots, t_n\} \cup \{t'_1, \dots, t'_n\}$ where*

- *$\nexists t \in cl \setminus \{t_1, \dots, t_n\}$ such that $t[RHS(fd)] = t_i[RHS(fd)]$ for some $i \in [1, n]$;*
- *let v be a value in $[\min_{t \in cl}(t[RHS(fd)]), \max_{t \in cl}(t[RHS(fd)])]$; then, $\forall i \in [1, n]$ the following conditions hold:*
  - *$t'_i[RHS(fd)] = v$;*
  - *$\forall A \in Attr(R)$ s.t. $A \neq RHS(fd)$, $t'_i[A] = t_i[A]$.*

Note that according to this definition, the set $\{t_1, \dots, t_n\}$ is required to be "maximal" in the sense that every time a tuple is in this set, the other tuples $t \in cl$ having the same value for $[RHS(fd)]$ must be included too.

The interval-based policy allows any tuple in a cluster to be replaced by a new tuple having a different value for attribute $RHS(fd)$.[1] For example, we may replace the values of the $Salary$ attribute of the tuples in cluster $\{t_1, t_2, t_3\}$ in Example 1 by a value equal to $73.33K$ (the mean of the three salary values for John). Or, if the reliability of sources

[1] Another kind of policy could use the interval $[\min_{t \in cl}(t[RHS(fd)]), \max_{t \in cl}(t[RHS(fd)])]$ in the new tuple, as the value for attribute $RHS(fd)$. In order to store, for each attribute, an appropriate interval, this kind of policy would require an extension of the database schema.

s_1, s_2, s_3 are 1, 3, and 2, respectively, we might replace the values of the $Salary$ attribute with the weighted mean $(70K*1+80K*3+70K*2)/6 = 75K$. Thus, the interval-based policy allows cases C3 and C4 in the Introduction to be handled.

We now show that Axiom A3 is satisfied by interval-based policies.

Theorem 2. *Let R be a relation, fd a functional dependency over R, and ξ_{fd} an interval-based policy for R w.r.t. fd. Then, for each $cl \in$ clusters(R,fd), it is the case that $|culprits(\xi_{fd}(cl), \{fd\})| < |culprits(cl, \{fd\})|$.*

Proof. Suppose R is a relation over the relational schema $S(A_1, \dots, A_n)$ and we have an fd $fd : A'_1, \dots, A'_k \to A'_{k+1}$ with $\{A'_1, \dots, A'_{k+1}\} \subseteq Attr(S)$ an FD over S. For $cl \in clusters(R, \{fd\})$, assume that the values $t[A'_{k+1}]$ of tuples $t \in cl$ are the union of single-value multi-sets $V_1, V_2, \dots, V_\ell$ (where every multi-set V_i contains the single value v_i with cardinality C_i). Before applying the policy, $|culprits(cl, \{fd\})| = \frac{(\sum_i C_i)^2 - \sum_i {C_i}^2}{2}$. By Definition 8, after applying an interval-based policy, the subset $\{t_1, \dots, t_n\}$ of cl is such that different multisets $V_{i_1}, \dots, V_{i_p}$ collapse into a single multiset V_a with cardinality $C_a = V_{i_1} + \cdots + V_{i_p}$. Hence, $|culprits(cl, \{fd\})|$ after a cluster simplification is

$$\frac{(\sum_i C_i)^2 - \sum_{i \neq i_1, \dots, i \neq i_p} {C_i}^2 - {C_a}^2}{2} = \frac{(\sum_i C_i)^2 - \sum_i {C_i}^2 - 2\sum_{i,j \in [i_1, \dots, i_p], i<j} C_i C_j}{2}$$

Thus, the number of culprits decreases by $\sum_{i,j \in [i_1, \dots, i_p], i<j} C_i C_j$. □

Finally, we (*i*) ensure that all members of the families of policies we defined satisfy our proposed axioms; (*ii*) characterize the relationships among the families; and (*iii*) ensure that all the kinds of IMPs we propose reduce the *dirtiness* or *degree of inconsistency* of a database according to the approaces proposed by several authors [Loz94, GH06, HK05, GH08] which focus on the logical structure of the inconsistency.

Proposition 1. *All members of the families of tuple-based, value-based, and interval-based policies satisfy Axioms A1, A2, A3, and A4.*

Observation 1. *Given a relation R over a schema S and a functional dependency fd : $A_1, \ldots, A_k \rightarrow B$ over R,*

- *for each tuple-based policy τ_{fd}, there is a value-based policy ν_{fd} such that $\tau_{fd}(R) \subseteq \nu_{fd}(R)$; moreover, if $Attr(S) = \{A_1, \ldots, A_k, B\}$, then $\tau_{fd}(R) = \nu_{fd}(R)$.*
- *for each value-based policy ν_{fd}, there is an interval-based policy ξ_{fd} such that $\nu_{fd}(R) = \xi_{fd}(R)$.*

Proposition 2. *Consider a relation R, a functional dependency fd over R, and an IMP γ_{fd} that is either a tuple-based, value-based, or interval-based policy. The dirtiness of $\gamma_{fd}(R)$ is less than or equal to the dirtiness of R for any of the definitions of dirtiness given in [Loz94, GH06, HK05, GH08].*

3.3.2 Multi-Dependency Policies

Suppose each *fd* $\in \mathcal{F}$ has a single-dependency policy associated with it (specifying how to manage the inconsistencies in the relation with respect to that FD). We assume that the system manager specifies a partial ordering $\leq_{\mathcal{F}}$ on the FDs, specifying their relative importance. Let $TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ be the set of all possible total orderings of FDs w.r.t. $\leq_{\mathcal{F}}$: this can be obtained by topological sorting.

Definition 9. *Given a relation R, a set of functional dependencies $\mathcal{F}$, a partial ordering $\leq_{\mathcal{F}}$, and an order $o = \langle fd_1, \ldots, fd_k \rangle \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$, a* multi-dependency IMP *(MDIMP for short) for R w.r.t. o and $\mathcal{F}$ is a function $\mu^o_{\mathcal{F}}$ from a relation R to a relation $\gamma_{fd_k}(\ldots \gamma_{fd_2}(\gamma_{fd_1}(R)) \ldots)$, where $\gamma_{fd_1}, \ldots, \gamma_{fd_k}$ are the singular dependency policies associated with $fd_1, \ldots, fd_k$, respectively.*

Basically, all that a total ordering does is to specify the order in which the conflicts are resolved. We start by resolving the conflict involving the first FD in the ordering, then the second, and so forth. However, different total orderings can lead to different results.

Example 3. *Consider the Salary Example presented in the Introduction and the set of FDs $\{fd_1,fd_2\}$ where fd_1 is Name $\rightarrow$ Salary and fd_2 is Name $\rightarrow$ Tax_bracket. Suppose the tuple-based policy τ_{fd_1} selects the tuple with the highest value of the Salary attribute (when inconsistency occurs), while τ_{fd_2} selects the lowest value of the Tax_bracket attribute. Under the total order $o = \langle fd_1,fd_2\rangle$, we get $\{(John, 80K, 20), (Mary, 90K, 30)\}$ as the result. Note that after τ_{fd_1} is applied, the other policy is not, because there is no further inconsistency w.r.t. fd_2. Therefore, τ_{fd_1} is solely responsible for deciding what tuples are part of the final answer. Under the total order $o = \langle fd_2,fd_1\rangle$, the result of applying the multi-dependency policy will be $\{(John, 70K, 15), (Mary, 90K, 30)\}$. Here, τ_{fd_2} decides which tuples are in the answer, causing the application of τ_{fd_1} to have no effect.*
Now consider the set of FDs $\{fd_1,fd_3\}$ where fd_3 is Salary $\rightarrow$ Tax_bracket, and suppose the value-based policy ν_{fd_1} states that, in case of inconsistency, the highest value for attribute Salary should be preferred, while ν_{fd_3} states that the lowest value for attribute Tax_bracket should be preferred. In this case, depending on which order we choose, the result of applying the multi-dependency policy will be: $\{(John, 80K, 15), (Mary, 90K, 30)\}$ (for $\langle fd_1,fd_3\rangle$), and $\{(John, 80K, 15), (John, 80K, 20), (Mary, 90K, 30)\}$ (for $\langle fd_3,fd_1\rangle$).

It is clear that the order in which violations of FDs get resolved plays an important role in determining the semantics of our system. One semantics assumes that the user or the system administrator somehow chooses a fixed total ordering rather than a partial ordering. This leads to the semantics specified in Definition 9. However, a natural question is whether we should say that a tuple is in the answer if it is present in the answer *irrespective* of which order is chosen. This is what we call the *Core* semantics below, and is analogous to cautious reasoning.

Definition 10. *Given a relation R, a set of functional dependencies $\mathcal{F}$ over R, and a partial ordering $\leq_{\mathcal{F}}$ on $\mathcal{F}$, the result of applying a policy under the* core semantics *is the set Core*$(R, \mathcal{F}, \leq_{\mathcal{F}}) = \bigcap\{\mu^{o}_{\mathcal{F}}(R) \mid o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})\}$.

Intuitively, the *Core* semantics looks at all total orderings compatible with the associated partial ordering on $\mathcal{F}$. If every such total ordering causes a tuple to be in the result (according to Definition 9), then the tuple is returned in the answer. Of course, one may also be interested in the following analogous "Possibility" problem.

Problem 1 (Possibility Problem)**.** *Given a relation R, a tuple $t \in R$, a set of functional dependencies $\mathcal{F}$ over R, and a partial ordering $\leq_{\mathcal{F}}$, does there exist a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^{o}_{\mathcal{F}}(R)$?*

We now state three complexity results.

Theorem 3. *Given a relation R, a set of functional dependencies $\mathcal{F}$, a partial order $\leq_{\mathcal{F}}$ over $\mathcal{F}$, and a tuple $t \in R$:*

1. *Determining whether $t \in$ Core$(R, \mathcal{F}, \leq_{\mathcal{F}})$ is coNP-complete.*

2. *Determining whether there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^{o}_{\mathcal{F}}(R)$ is NP-complete.*

3. *If the arity of R is bounded, then the complexity of the problems (1) and (2) above is in PTIME.*

Proof. **Proof of Theorem 3**

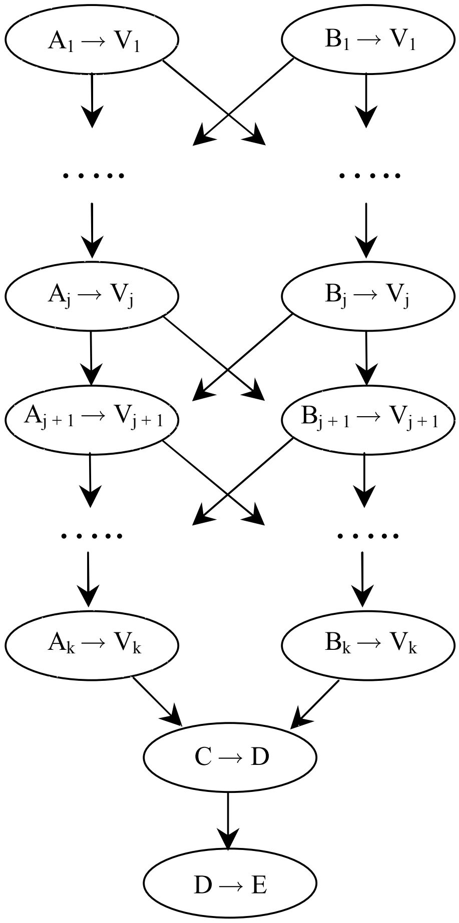


Figure 3.1: Partial order $\leq_{\mathcal{F}}$ for relational schema S.

Statement 2. (*Membership*) A polynomial size witness for this problem is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^o_{\mathcal{F}}(R)$. As any single FD policy can be computed in polynomial time, this witness can be verified in polynomial time by applying the policies one at a time, according to o, and finally checking whether $t \in \mu^o_{\mathcal{F}}(R)$.

(*Hardness*) We show a LOGSPACE reduction from 3SAT [Pap94]. An instance of 3SAT is a pair $\langle U, \Phi \rangle$, where $U = \{P_1, P_2, \ldots, P_k\}$ is a set of propositional variables and Φ is a propositional formula of the form $C_1 \wedge \cdots \wedge C_n$ defined over U. Specifically, each C_i (with $1 \leq i \leq n$) is a clause containing exactly three (possibly negated) propositional variables in U.

We show how Φ can be encoded by an instance $\langle R, \mathcal{F}, \leq_{\mathcal{F}}, t' \rangle$ of our problem. Let S be the relational schema $S(A_1, B_1, V_1, \ldots, A_k, B_k, V_k, C, D, E)$, where attributes A_j, B_j, V_j correspond to variable P_j with $j \in [1..k]$, and C, D, E are extra attribtues.

The set of functional dependencies $\mathcal{F}$ for S is $\{fd_{A,j} : A_j \rightarrow V_j, fd_{B,j} : B_j \rightarrow V_j \mid j \in [1..k]\} \cup \{fd_C : C \rightarrow D, fd_D : D \rightarrow E\}$. Consider the following tuple-based total policies associated with the FDs in $\mathcal{F}$: $\gamma_{fd_{A,j}}$ stating *choose the highest value of* V_j *for each cluster*, $\gamma_{fd_{B,j}}$ stating to *choose the lowest value of* V_j *for each cluster* (with $j \in [1..k]$), γ_{fd_C} stating to *delete the whole set of inconsistent tuples in each cluster*, and γ_{fd_D} stating to *delete the whole set of inconsistent tuples in each cluster*. The partial order for $\mathcal{F}$ is defined as follows: $\forall j \in [1..k-1]$ and $Y \in \{A, B\}$, $fd_{Y,j} < fd_{Y,j+1}$ and $fd_{Y,k} < fd_C$, and $fd_C < fd_D$. The partial order is ilustrated in Figure 3.1; note that for each variable P_j only the precedence between $fd_{A,j}$ and $fd_{B,j}$ are not specified.

Let R be an instance of S defined as follows. Initially R is empty. Then, for each $P_j \in U$ and for each $C_i \in \Phi$,

- if making P_j *true* makes C_i *true* we add to R the tuple t such that $t[A_j] = t[B_j] = p_j$, $t[V_j] = 1$, $t[C] = c_i$, $t[D] = t[E] = 1$, and for each $X \notin \mathit{Attr}(S) \setminus \{A_j, B_j, V_j, C, D, E\}$, $t[X] = k_1$ where k_1 is a new symbol and p_j and c_i are symbols that represent variable P_j and clause C_i, respectively;

- if making P_j *false* makes C_i *true* we add to R the tuple t such that $t[A_j] = t[B_j] = p_j$, $t[V_j] = 0$, $t[C] = c_i$, $t[D] = t[E] = 1$, and for each $X \in \mathit{Attr}(S) \setminus \{A_j, B_j, V_j, C, D, E\}$, $t[X] = k_1$.

Moreover, for each $C_i \in \Phi$ we add to R the tuple t such that $t[C] = c_i$, $t[D] = t[E] = 2$, and for each $X \in \mathit{Attr}(S) \setminus \{C, D, E\}$, $t[X] = k_2$ where k_2 is new symbol. Finally, R also contains the tuple t' such that $t'[D] = 2$, $t'[E] = 3$ and for each $X \in Attr(S) \setminus \{D, E\}$, $t'[X] = k_3$ where k_3 is new symbol.

We now prove that Φ is satisfiable iff there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t' \in \mu^o_{\mathcal{F}}(R)$.

($\Rightarrow$) Assume that Φ is satisfiable, we must show that there exists a total order $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t' \in \mu^o_{\mathcal{F}}(R)$.

The total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ is obtained as follows. Let $U' \subseteq U$ be the set of propositional variables made *true* by a satisfying assignment for Φ. For each $P_j \in U'$, o requires that $fd_{A,j} < fd_{B,j}$; this means that for the tuples t such that $t[A_j] = t[B_j] = p_j$, the value $t[V_j] = 1$ is chosen by $\gamma_{fd_{A,j}}$, and that $\gamma_{fd_{B,j}}$ will not have any effect on R. For each $P_j \in U \setminus U'$ (the variables that are assigned *false* by the satisfiable assignment), o requires that $fd_{B,j} < fd_{A,j}$; this means that for the tuples t such that $t[A_j] = t[B_j] = p_j$, the value $t[V_j] = 0$ is chosen by $\gamma_{fd_{B,j}}$, and that $\gamma_{fd_{A,j}}$ will not have any effect on R. This gives an order between each $fd_{A,j}$ $fd_{B,j}$, and that is enough to define a total ordering o according to the partial ordering $\leq_{\mathcal{F}}$, since the ordering for the other FDs is already defined by $\leq_{\mathcal{F}}$.

Let R_1 be the relation resulting from the application of the policies associated to the FDs $fd_{A,j}$ and $fd_{B,j}$ (with $j \in [1..k]$) according with the above-specified order. At this point, column C contains values for each of the clauses that are made true by the assignment, and since this is a satisfiable assignment for Φ, it must be the case that all clauses in Φ can be made true. Therefore, it has to be the case that $\pi_C(R_1) = \{c_1, \ldots, c_n\}$. Moreover, since for each $C_i \in \Phi$, the relation R_1 also contains a tuple t such that $t[C] = c_i$ and $t[D] = 2$, there are n clusters w.r.t. fd_C (one for each C_i). Order o states that γ_{fd_C} musr be applied to R_1, and then each of these cluster is deleted (according to the policy defined by γ_{fd_C}); let relation R_2 be the result of doing that. Therefore, the only tuple which remains in R_2 is t'. Finally, the application of the last policy γ_{fd_D} does not have any effect (since there are no inconsistent tuples w.r.t. fd_D), and t' belongs to $\mu^o_{\mathcal{F}}(R)$.

($\Leftarrow$) Assume now that there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t' \in \mu^o_{\mathcal{F}}(R)$. According to the partial ordering $\leq_{\mathcal{F}}$, γ_{fd_D} must be the last policy applied to the relation. In order for t' to be part of $TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ it has to be the case that after applying all the other policies there is no cluster w.r.t. fd_D (otherwise γ_{fd_D} would have deleted the whole cluster including t').

The fact that there are no conflicting tuples in $\mu^o_{\mathcal{F}}(R)$ w.r.t. fd_D entails that there is no tuple $t \in \mu^o_{\mathcal{F}}(R)$ such that $t[D] = 2$ and $t[E] \neq 3$. Therefore, all the tuples t such that $t[C] = c_i$ and $t[D] = t[E] = 2$ must have been deleted by γ_{fd_C}, and this can happen only if there was at least a cluster for each C_i. Let R_1 be the relation obtained after applying all the policies associated with the FDs $fd_{A,j}$ and $fd_{B,j}$ (with $j \in [1..k]$) according to o. R_1 contains for each $C_i \in \Phi$, a tuple t such that $t[C] = c_i$. It is important to note that, with respect to the assignment of truth values for variables in Φ, this means that it is possible to make each C_i true, and therefore, Φ is satisfiable.

The satisfying assignment for Φ is obtained from R_1 in the following way. Note that, for each variable P_j the set $\pi_{V_j}(\sigma_{A_j=p_j}(R_1))$ is a singleton, either $\{0\}$ or $\{1\}$; this is because no matter in which order $fd_{A,j}$ and $fd_{B,j}$ were applied, o ensures that either all 1's or all 0's were deleted for each P_j. Therefore, for each variable P_j, if $\pi_{V_j}(\sigma_{A_j=p_j}(R_1)) = \{1\}$ then P_j is assigned the truth value *true*, otherwise (i.e., $\pi_{V_j}(\sigma_{A_j=p_j}(R_1)) = \{0\}$) P_j is assigned *false*.

Statement 1. (*Membership*) A polynomial size witness for the complement of this problem is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \notin \mu^o_{\mathcal{F}}(R)$. As any single FD policy can be computed in polynomial time, this witness can be verified in polynomial time by applying the policies one at a time, according to o, and finally checking whether $t \notin \mu^{o}_{\mathcal{F}}(R)$.

(*Hardness*) The complement of the problem of determining whether tuple $t \in$ *Core*$(R, \mathcal{F}, \leq_{\mathcal{F}})$ is the problem of deciding whether there is a total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \notin \mu^{o}_{\mathcal{F}}(R)$. We show a LOGSPACE reduction from the Possibility problem to the complement of our problem.

Let $\langle R_1, \mathcal{F}_1, \leq_{\mathcal{F}_1}, t_1\rangle$ be an instance of the problem of deciding whether there is a total ordering $o_1 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_1)$ such that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$. We define an instance $\langle R_2, \mathcal{F}_2, \leq_{\mathcal{F}_2}, t_2\rangle$ of our problem as follows.

Given the relational schema $S_1(A_1, \ldots, A_n)$ of R_1, we define the relational schema S_2 of R_2 as $S_2(A_1, \ldots, A_n, B, C)$. Let R_2 be initially empty. For each tuple $t \in R_1 \setminus \{t_1\}$ we add to R_2 the tuple t' such that $t'[X] = t[X]\ \forall X \in Attr(S_1)$ and $t'[B] = t'[C] = k_1$, where k_1 is a new symbol. Moreover, we add to R_2 the following tuples:

- t_1^* such that $\forall X \in Attr(S_1)$, $t_1^*[X] = t_1[X]$, and $t_1^*[B] = k_2$, where k_2 is a new symbol, and $t_1^*[C] = 0$.

- t_2 such that $\forall X \in Attr(S_1)$, $t_2[X] = k_3$, where k_3 is a new symbol, $t_2[B] = k_2$, and $t_2[C] = 1$.

Let $\mathcal{F}_2$ be $\mathcal{F}_1 \cup \{fd : B \rightarrow C\}$, γ_{fd} be a tuple-based total policy stating that *the lowest value of* C *must be chosen*, and $\leq_{\mathcal{F}_2}$ be the partial order defined by adding, $\forall fd' \in \mathcal{F}_1$, $fd' < fd$ to $\leq_{\mathcal{F}_2}$.

We now prove that there is $o_1 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_1)$ such that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$ iff there is $o_2 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_2)$ such that $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$.

($\Rightarrow$) Assume that there is a total ordering $o_1 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_1)$ such that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$. We can define $o_2 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_2)$ such that $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$ as follows: o_2 is equal to o_1

plus $fd' < fd$ where fd' is the last FD in o_1. The fact that $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$ implies that the tuple $t_1^* \in R_2$ will be in $\mu^{o_1}_{\mathcal{F}_2}(R_2)$. Thus, as $t_2[B] = t_1^*[B]$ and $t_2[C] > t_1^*[C]$ the policy γ_{fd} deletes t_2 form R_2. Hence, $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$.

($\Leftarrow$) Assume now that there is a total ordering $o_2 \in TOT_{\leq_{\mathcal{F}_1}}(\mathcal{F}_2)$ such that $t_2 \notin \mu^{o_2}_{\mathcal{F}_2}(R_2)$. As only γ_{fd} can delete t_2, this implies that before applying γ_{fd} the tuple t_1^* was in the result of $\mu^{o_1}_{\mathcal{F}_2}(R_2)$ (where o_1 is equal to o_2 except the ordering relationships involving fd). Hence, $t_1 \in \mu^{o_1}_{\mathcal{F}_1}(R_1)$.

Statement 3. Assuming that the arity of R is bounded by a constant b, the cardinality of $\mathcal{F}$ is bounded by 2^b, and the number of possible ordering in $TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ is bounded by the factorial of 2^b, which is still a constant w.r.t. the cardinality of R. Thus, since any single FD policy can be computed in polynomial time, checking whether there is total ordering $o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ such that $t \in \mu^{o}_{\mathcal{F}}(R)$ (or equivalently $t \notin \mu^{o}_{\mathcal{F}}(R)$) and determining whether $t \in \mathit{Core}(R, \mathcal{F}, \leq_{\mathcal{F}})$ are in PTIME. □

Basically, the source of complexity is the fact that there may be exponentially many total orderings in $TOT_{\leq_{\mathcal{F}}}(\mathcal{F})$ induced by a given partial ordering $\leq_{\mathcal{F}}$ on $\mathcal{F}$. However, if the arity of R is bounded by a constant b, the number of such total orderings is bounded by a constant as well, leading to the PTIME result. [2]

We do not specify a possible semantics which returns $\bigcup\{\mu^{o}_{\mathcal{F}}(R) \,|\, o \in TOT_{\leq_{\mathcal{F}}}(\mathcal{F})\}$, since this can yield a relation with sources of inconsistency that were not present before the application of the multi-dependency policy, violating in this way Axiom A3. In the following, we show an example of how such a situation can arise.

Example 4. *Consider the following relation R:*

[2] It should be noted that we assume that policies can be computed in polynomial time. We do not consider NP-hard policies such as, *i.e.*, among a set V of inconsistent (possibly negative) values choose a nonempty subset $V' \subset V$ such that $\sum_{v \in V'} v = 0$.

	Name	*Salary*	*Tax_bracket*
t_1	$John$	$70K$	*15*
t_2	$John$	$80K$	*20*

Let fd$_1$ *be Name* → *Salary, and fd*$_2$ *be Salary* → *Tax_bracket. Suppose we have two interval-based policies* ξ_{fd_1} *and* ξ_{fd_2}*, both stating that conflicting values must be replaced by their mean. Assuming that fd*$_1$ *and fd*$_2$ *are incomparable w.r.t.* $\leq_{\mathcal{F}}$*, then there are two possible total orders:* ⟨*fd*$_1$*,fd*$_2$⟩ *and* ⟨*fd*$_2$*,fd*$_1$⟩*. In the first case, the result of applying the corresponding multi-dependency policy is* $R' = \{(\mathit{John}, 75K, 17.5)\}$*, whereas in the second case the result is* $R'' = \{(\mathit{John}, 75K, 15), (\mathit{John}, 75K, 20)\}$*. It is easy to see that* $|culprits(R' \cup R'', \{fd_1, fd_2\})| > |culprits(R, \{fd_1, fd_2\})|$.

3.4 Specifying IMPs

The general characterization of IMPs provided in Definition 4 is highly expressive and allows IMPs to specify very complex policies. In this section, we suggest possible options for languages within which IMPs can be expressed.

IMPs can be viewed as a set of rules that a user specifies in order to manage inconsistency with respect to sets of constraints. One specific approach towards designing a policy specification language is to define these rules as *logic programs* [Llo87], which provide clear and well-studied semantics. The relational instances may be represented in a first-order language where the knowledge base consists of tuples and the inconsistency structures (culprits and clusters) of the relation.

We assume standard logic programming notation, and in particular we will refer to constants in the different domains with lowercase letters, whereas we use uppercase letters for variables. Let R be a relation over schema $S(A_1, \ldots, A_n)$, and let $\mathcal{F}$ be a set of functional dependencies over S. We assume the existence of an $(n+1)$-ary predicate symbol $tuple\_R$ such that for each tuple $(a_1, \ldots, a_n) \in R$, the logic program Δ_R contains the fact $tuple\_R(id, a_1, \ldots, a_n)$, where id is a number that uniquely identifies tuple $(a_1, \ldots, a_n)$ in R. Moreover, *cluster* is a 3-ary predicate symbol. Let $c \in$ *clusters*$(R, \{$*fd*$\})$ where *fd* $\in \mathcal{F}$; for each tuple $(id, a_1, \ldots, a_n) \in c$, the logic program Δ_R contains the fact *cluster*$(id, c,$*fd*$)$.

Example 5. *Consider relation* Emp *from Example 1, where* $\mathcal{F} = \{$*fd* $: Name \rightarrow Salary\}$*;* Δ_R *contains the following facts:*

```
tuple_Emp(1, john, 70, 15).
tuple_Emp(2, john, 80, 20).
tuple_Emp(3, john, 70, 25).
tuple_Emp(4, mary, 90, 30).
cluster(1,1,fd).
cluster(2,1,fd).
cluster(3,1,fd).
```

An IMP may thus be simply described as a logic program that will be applied over the knowledge base Δ_R, and whose unique least model corresponds to $\gamma(R)$. To this end, given a policy γ_{fd}, we might use an $(n+1)$-ary predicate symbol *result_γ_fd*. Intuitively, *result_γ_*$fd(id, a_1, \ldots, a_n)$ is true if and only if $\gamma_{fd}(R)$ contains tuple $(a_1, \ldots, a_n)$.[3]

Example 6. *Suppose a user specifies policy PolMin_fd for relation* Emp*; PolMin_fd indicates that all values for attribute* Salary *of tuples within a cluster w.r.t.* $Name \rightarrow Salary$ *should be changed to the minimum value for* Salary *among all tuples in the cluster. The following logic program* Π_{PolMin_fd} *describes policy PolMin_fd. For this example we as-*

[3] Observe that we are assuming that policies are being specified using Prolog programs; other semantics for logic programs, such as Answer Set semantics or well-founded semantics could also be adopted.

sume the existence of predicate min*, such that min*(C, X, V) *is true if and only if value* V *is the minimum value for attribute* X *in a cluster with id* C*.*

```
result_PolMin_fd(ID, Name, SalaryMin, _) <--
                          tuple_Emp(ID, Name, Salary, _),
                          cluster(ID, C, fd),
                          min(C, salary, SalaryMin).
```

Logic programs are a powerful formalism to express IMPs. If such a language were to be implemented, an interesting problem would be that of checking whether a given program corresponds to a valid IMP, *i.e.*, identify the circumstances under which logic programs satisfy Axioms A1 through A4 of Definition 4.

Another possible option is that of declaring IMPs as SQL stored procedures. Most DBMSs provide a powerful procedural language that can be used to define procedures and functions. A policy specified in this way can be implemented for a particular functional dependency, or more general parametric procedures can be defined that take a functional dependency as a parameter. For instance, the user could specify a policy that, for each cluster, deletes every tuple whose value for the right-hand side attribute is not the minimum of the cluster; the policy can be implemented generically to take any FD of the form $X \rightarrow Y$, where X is a list of attributes and Y a single attribute.

Moreover, appropriate extensions to SQL are needed to support the specification of IMPs, in order to allow the user to:

- *Associate a functional dependency with a relation.* SQL does not provide an easy way to specify functional dependencies; one possible syntactic extension to the language to allow this could work in the same way a key constraint (or primary key) is added to a relation.

- *Associate a policy with a relation and a functional dependency*, *i.e.*, specify the stored procedure that implements the policy and the corresponding constraint (for instance, a statement of the form

  ```
  ALTER TABLE Emp ADD POLICY  P1 REFERENCES fd
  ```

  could be used to associate policy P1 with relation *Emp* w.r.t. functional dependency *fd*).

- *Indicate what policy should be used in a query, and the order of application with respect to relational operators*. IMPs are designed to be usable in conjunction with relational algebra operators (the relationships between IMPs and relational operators will be studied in Section 3.7). When issuing a query, the user may want to specify that a certain policy should be applied as part of the query, and whether the policy or the relational operators are applied first. For instance, a query of the form:

  ```
      SELECT * FROM Emp WHERE
           Name = 'John' USING POLICY P1 FIRST
  ```

  asks for the set of tuples whose value for attribute *Name* is *John* and specifies that policy P1 should be applied before the selection operator.

- *Specify the semantics in the presence of multiple policies*. Additional SQL extensions should be used in order to express the semantics of the application of multiple policies. For instance, a query of the form:

  ```
      SELECT * FROM Emp WHERE
           Name = 'John' USING POLICY P1, P2 LAST CORE
  ```

could state that after applying the selection operator, both policy $P1$ and $P2$ must be applied under the core semantics.

Finally, for the cases where users are not familiar with (declarative or imperative) programming, a simplified view of how policies are specified could be provided. For instance, a simple and user-friendly graphical interface could allow the user to specify conditions under which tuples should be kept or deleted (in the case of tuple-based policies), or input functions that will generate the new values for the right-hand side of functional dependencies in the case of value- or interval-based policies. This allows users to effectively communicate how they want their data to be manipulated without having to worry about how the policies will be internally represented and implemented.

3.5 Relationship with belief change operators

An important area of research related to inconsistency management is that of *belief change* to belief sets (sets of formulas closed under consequence) and belief bases (sets of formulas not necessarily closed under consequence), as discussed in Section 2.1.1. It seems reasonable to think that inconsistency management techniques in relational databases are materializations of some variations of belief change methods. This is true for some of the methods proposed by [ABC99, Cho07, BFFR05], but the relationship w.r.t. IMPs is less clear. The main goal of belief change frameworks is to maintain consistency while contracting or revising belief systems. This is the fundamental difference with the IMPs framework since, by design, policies can be defined that do not remove inconsistency completely. However, the two approaches have a lot in common and it is interesting to study their differences and similarities. In any practical database application, only belief bases are relevant, and hence, in this chapter, we briefly discuss relationships between

IMPs and axioms for updating belief bases [Han93, Han97] as opposed to axioms to update belief sets [AGM85, Gar88b]. Given an IMP based on a single functional dependency, we first show how to define an associated revision operator.

Definition 11. *Let R be a relation over relational schema S, fd be an FD over S, and let γ_{fd} be any IMP for R w.r.t. fd. Let K_R be the first-order belief base obtained from R by treating the tuples in R as ground atoms and the FD as a logical formula in the obvious way.*[4] *We say that* $\dot{+}_{\gamma_{fd}}$ is a belief revision operator that corresponds to γ_{fd} *iff:*

- *for each tuple $t \in \gamma_{fd}(R)$ there exists a sentence $\alpha_t \in K_R \dot{+}_{\gamma_{fd}}$ fd, such that α_t is the first-order encoding of t,*
- *for each sentence $\alpha \in K_R \dot{+}_{\gamma_{fd}}$ fd either there exists a tuple $t \in \gamma_{fd}(R)$ such that α_t is the first-order encoding of t, or $\alpha =$ fd, and*
- *$\gamma_{fd}(R)$ is consistent w.r.t. fd iff fd $\in K_R \dot{+}_{\gamma_{fd}}$ fd.*

Intuitively, $\dot{+}_{\gamma_{fd}}$ is a revision operator in the sense of [Han93] that implements γ_{fd}. [Han93] proposes the satisfaction of four axioms for belief bases revision operators $\oplus$. These axioms are:

- *Success.* $\alpha \in K \oplus \alpha$.
- *Inclusion.* $K \oplus \alpha \subseteq K \cup \alpha$.
- *Relevance.* If $\beta \in K$ and $\beta \notin K \oplus \alpha$, then there is a set K' such that $K \oplus \alpha \subseteq K' \subseteq K \cup \{\alpha\}$ such that K' is consistent but $K' \cup \{\beta\}$ is inconsistent.

[4] K_R contains the atom $R(\vec{t})$ for each tuple $\vec{t} \in R$. In addition, as described by [ABC$^+$03b], if $X \rightarrow Y$ is an FD over relation P such that X is the set of attributes corresponding to variables $\vec{x}$ and Y is the set of attributes corresponding to variables $\vec{y}$, then *fd* can be expressed as the formula: $\forall \vec{x}, \vec{y}, \vec{z}, \vec{y'}, \vec{z'}.(\neg P(\vec{x}, \vec{y}, \vec{z}) \vee \neg P(\vec{x}, \vec{y'}, \vec{z'}) \vee \vec{y} = \vec{y'})$.

- *Uniformity.* If it holds for all subsets K' of K that $K' \cup \alpha$ is inconsistent if and only if $K' \cup \beta$ is inconsistent, then $K \cap (K \oplus \alpha) = K \cap (K \oplus \beta)$.

The result below specifies when the belief revision operator $\dot{+}_{\gamma_{fd}}$ corresponding to an IMP γ_{fd} satisfies the Success axiom.

Theorem 4. *Let R be a relation over the relational schema S, let fd be the only functional dependency over S, and let γ_{fd} be an IMP for R w.r.t. fd. If K_R is the first-order belief base obtained from R then: $\dot{+}_{\gamma_{fd}}$ satisfies the* Success *axiom iff $|culprits(\gamma_{fd}(R), \{fd\})| = 0$,* i.e.*, the application of γ_{fd} over R removes all the inconsistency in R w.r.t. fd.*

Proof. Operator $\dot{+}_{\gamma_{fd}}$ satisfies success iff $fd \in K_R \dot{+}_{\gamma_{fd}} fd$, which by definition of $\dot{+}_{\gamma_{fd}}$ means that $\gamma_{fd}(R)$ is consistent w.r.t. *fd*, which is true iff $|culprits(\gamma_{fd}(R), fd)| = 0$. □

The result below specifies when the belief revision operator $\dot{+}_{\gamma_{fd}}$ corresponding to an IMP γ_{fd} satisfies the Inclusion axiom.

Theorem 5. *Let R be a relation over the relational schema S, let fd be the only functional dependency over S, let γ_{fd} be an IMP for R w.r.t. fd, and let K_R be the first-order belief base obtained from R; operator $\dot{+}_{\gamma_{fd}}$ satisfies the* Inclusion *axiom iff γ_{fd} is a tuple-based policy.*

Proof. ($\Rightarrow$) If $\dot{+}_{\gamma_{fd}}$ satisfies inclusion then $K_R \dot{+}_{\gamma_{fd}} fd \subseteq K_R \cup fd$. If $fd \in K_R \dot{+}_{\gamma_{fd}} fd$ then $K_R \dot{+}_{\gamma_{fd}} fd = K' \cup \{fd\}$ and therefore $K' \subseteq K_R$ and since $\gamma_{fd}(R)$ is effectively the relational instance of K' we can conclude that $\gamma_{fd}(R) \subseteq R$, therefore $\gamma_{fd}(R)$ is a tuple-based policy. On the other hand, if $fd \notin K_R \dot{+}_{\gamma_{fd}} fd$ then $K_R \dot{+}_{\gamma_{fd}} fd \subseteq K_R$. Since $\gamma_{fd}(R)$ is effectively the relational instance of $K_R \dot{+}_{\gamma_{fd}} fd$ we can conclude that $\gamma_{fd}(R) \subseteq R$, therefore $\gamma_{fd}(R)$ is a tuple-based policy.

($\Leftarrow$) If $\gamma_{fd}(R)$ is a tuple-based policy, then by definition $\gamma_{fd}(R) \subseteq R \subseteq R \cup \{fd\}$. Let $\dot{+}_{\gamma_{fd}}$ be the revision operator that corresponds to $\gamma_{fd}(R)$ we have that $K_R \dot{+}_{\gamma_{fd}} fd \subseteq K_R \cup fd$. Therefore, $\dot{+}_{\gamma_{fd}}$ satisfies inclusion. □

Observation 2. *The belief revision operator $\dot{+}_{\gamma_{fd}}$ corresponding to any IMP γ_{fd} is not guaranteed to satisfy the Relevance axiom.*

The Relevance axiom was introduced by Hansson in order to require minimum loss of information in the revision process. In this sense this axiom ensures that the sentences that are directly in conflict with the epistemic input are eliminated. In our approach, IMPs are defined so users can apply any criterion for resolving inconsistency, including but not restricted to minimum information loss. For instance, in Example 1 a user could decide that sources s_2 and s_3 are not trustworthy and apply a policy that deletes both tuples t_2 and t_3. This is a valid IMP but it does not satisfy relevance: tuple t_3 is removed even though it is not directly in conflict with tuple t_1, the one that remains in the knowledge base. A weaker version of this axiom was introduced by Hansson [Han97] later on for non-prioritized revision:

Core Retainment. If $\beta \in K$ and $\beta \notin K \oplus \alpha$, then there is a set K' such that $K' \subseteq K \cup \{\alpha\}$ such that K' is consistent but $K' \cup \{\beta\}$ is inconsistent.

Theorem 6. *Let R be a relation over the relational schema S, let fd be the only functional dependency over S, let γ_{fd} be a tuple-based IMP for R w.r.t. fd, and let K_R be the first-order belief base obtained from R. Then, $\dot{+}_{\gamma_{fd}}$ satisfies* Core Retainment.

Proof. Let t be a tuple in R that is not in $\gamma_{fd}(R)$. As $t \in R$ and $t \notin \gamma_{fd}(R)$, there is $c \in \mathit{culprits}(R, fd)$ such that $t \in c$ (see Axiom A1 from Definition 4). Let t' be a tuple in R distinct from t and such that the culprit $\{t, t'\} \in c$. Suppose that β is the sentence representing tuple t, and K' consists of the sentence representing t' and that representing

fd. Clearly, $\beta \in K_R$ and $\beta \notin K_R \dot{+}_{\gamma_{fd}} fd$. Moreover, $K' \subseteq K_R \cup \{fd\}$ and K' is consistent but $K' \cup \{\beta\}$ is inconsistent. $\Box$

Finally, we note that the *Uniformity* postulate holds trivially because the "if" part is equivalent to saying that $\alpha = fd$ is equivalent to $\beta = fd'$. It is reasonable to assume that if there exists $fd' \in \mathcal{F}$ such that fd' is logically equivalent to fd, then they are exactly the same functional dependency; therefore, as operator $\dot{+}_{\gamma_{fd}}$ is defined exclusively for fd we can conclude that fd and fd' have the same associated policy.

3.6 Relationship with preference-based approaches in Consistent Query Answering

In the last few years, a great deal of attention has been devoted by the databases community to the problem of extracting reliable information from data inconsistent w.r.t. integrity constraints. Most work dealing with this problem is based on the notions of *maximal consistent subsets* introduced originally by [FUV83, FKUV86] as "flocks" in the context of database updating, and later studied as maximal consistent subsets for integrating multiple knowledge bases [BKM91, BKMS91], and then defined as "repairs" of databases and *consistent query answers* (CQA) introduced in [ABC99]. A repair of an inconsistent database is a new database, on the same schema as the original database, satisfying the given integrity constraints and that is "minimally" different from the original database (the minimality criterion aims at preserving the information in the original database as much as possible). Thus, an answer to a given query posed to an inconsistent database is said to be consistent if the same answer is obtained from every possible repair of the database. Even though several works investigated the problem of repairing and querying inconsistent data considering different classes of queries and constraints, only recently there have been two proposals which shifted attention towards improving the quality of consistent answers. These approaches developed more specific repairing strategies that reduce the number of possible repairs to be considered and improve their quality according to some criteria specified by the database administrator on the basis of users' preferences. We will analyze the relationships between IMPs and each of the proposals in turn.

3.6.1 Active Integrity Constraints

Active Integrity Constraints (AICs for short) are an extension of integrity constraints for consistent database management introduced in [CGZ09]. Repairs in this work are defined as minimal sets (under inclusion) of update actions (tuple deletions/insertions) and AICs specify the set of update actions that are used to restore data consistency. Hence, among the set of all possible repairs, only the subset of *founded repairs* consisting of update actions supported by AICs is considered.

An AIC is a production rule where the body is a conjunction of literals, which should be false for the database to be consistent, whereas the head is a disjunction of *update atoms* that have to be performed if the body is true (that is the constraint is violated). As an example, consider the relation Emp of Example 1 with the FD *fd* : *Name* → *Salary*. The following AIC specifies that if the FD is violated, then the tuple with the highest salary has to be removed: $\forall N, S, S', T, T'[Emp(N, S, T), Emp(N, S', T'), S < S' \rightarrow -Emp(N, S', T')]$. In this case, among the set of possible repairs of relation Emp w.r.t. fd which delete one of the conflicting tuples to restore data consistency, only founded repairs deleting the tuple with the highest salary are considered.

Even though AICs are defined for a wider range of integrity constraints (universally quantified and general integrity constraints), while IMPs are only defined for functional dependencies, if we restrict our analysis to functional dependencies we can state the relationship between founded repairs and IMPs.

Let fr be a founded repair for the relation R w.r.t. a given set of AICs. The relation which results by performing the update actions in fr on R is denoted $R \circ fr$.

Theorem 7. *Let R be a relation over the relational schema $S(A_1, \ldots, A_n)$ and fd a functional dependency over S. W.l.o.g., assume that fd is of the form $A_1, \ldots, A_k \rightarrow A_{k+1}$, where $k + 1 \leq n$. Let μ be the AIC of the form*

$$\begin{aligned} \forall X \quad [\, & S(X_1, \ldots, X_k, X_{k+1}, \ldots, X_n), S(X_1, \ldots, X_k, X'_{k+1}, \ldots, X'_n), \\ & X_{k+1} \neq X'_{k+1}, \varphi(X_0) \rightarrow -S(X_1, \ldots, X_k, X'_{k+1}, \ldots, X'_n)\,] \end{aligned}$$

where $X = \bigcup_{i=1}^{k} X_i \cup \bigcup_{i=k+1}^{n}(X_i \cup X'_i)$ and $\varphi(X_0)$ is a conjunction of built-in atoms with $X_0 \subseteq X$. For each founded repair fr for R w.r.t. μ, there exists a tuple-based IMP γ_{fd} such that $R \circ fr = \gamma_{fd}(R)$.

Proof. Let $fr = \{-\alpha_1, \ldots, -\alpha_k\}$ be an arbitrary founded repair for R w.r.t. fd. We define γ_{fd} as a tuple-based policy that for each update action $-\alpha_i$ in fr, with $1 \leq i \leq k$, deletes the tuple α_i from R. Clearly, $R \circ fr = \gamma_{fd}(R)$, since each $-\alpha_i \in fr$ indicates that tuple α_i is deleted from R when fr is applied, and $\gamma_{fd}(R)$ deletes from R exactly the tuples α_i for which the update action $-\alpha_i$ appears in fr.

We now show that γ_{fd} satisfies the axioms from Definition 4. Let α_i be an arbitrary tuple in R that is not in $\gamma_{fd}(R)$. By definition, $\gamma_{fd}(R)$ only deletes a tuple α_i from R if $-\alpha_i \in fr$; if this is the case, then $-\alpha_i$ is supported by μ, that is there is a substitution σ of variables in X with constants of R such that

- $\alpha_i = S(\sigma(X_1), \ldots, \sigma(X_k), \sigma(X'_{k+1}), \ldots, \sigma(X'_n))$, and
- $t = S(\sigma(X_1), \ldots, \sigma(X_k), \sigma(X_{k+1}), \ldots, \sigma(X_n))$, with $\alpha_i, t \in R$, and
- $\sigma(X_{k+1}) \neq \sigma(X'_{k+1})$ and $\varphi(\sigma(X_0))$ evaluates to true.

This means that α_i is involved in a conflict w.r.t. *fd*, *i.e.*, it belongs to a culprit and thus Axiom A1 is satisfied. Given that $\gamma_{\mathit{fd}}(R)$ only deletes tuples from R, then we have that $\gamma_{\mathit{fd}}(R) \subseteq R$, and therefore Axiom A2 and Axiom A4 are satisfied. Lastly, as each $\alpha_i \in R$ that is not in $\gamma_{\mathit{fd}}(R)$ belongs to a culprit w.r.t. R and *fd*, removing α_i from R results in at least one culprit disappearing, and therefore Axiom A3 is satisfied as well. □

Moreover, there are tuple-based policies that are not expressible by AICs.

Observation 3. *Given a relation R with the functional dependency fd, there are tuple-based policies γ_{fd} for which there is no AIC μ (which extends fd) such that a founded repair for R w.r.t. μ is equal to $\gamma_{\mathit{fd}}(R)$; this fact holds even if $\gamma_{\mathit{fd}}(R)$ is consistent w.r.t. fd.*

For instance, consider the FD *Name* → *Salary* from Example 1 and an associated tuple-based policy stating that every tuple except the ones containing the average value for *Salary* in each cluster should be deleted; this is not expressible by AICs. Basically, this limitation of AICs follows from the fact that aggregates on clusters cannot be expressed in general.

Observation 4. *Neither value- nor interval-based policies are expressible as update actions of AICs.*

Value-based and interval-based policies could be implemented as a series of deletion and insertion of tuples, *e.g.*, modifying the value for *Salary* of tuple t_2 in the Emp relation of Example 1 from $80K$ to $70K$ so that *fd* : *Name* → *Salary* is satisfied can be done with the update actions $-$*Emp*$($*John*$, 80K, 20)$ and $+$*Emp*$($*John*$, 70K, 20)$. However, there is no repair for the relation *Emp* w.r.t. fd consisting of both these update actions since deleting t_2 is already minimal (under inclusion) for restoring consistency. As every founded repair is a repair, there is no founded repair consisting of both these update actions.

3.6.2 Prioritized Repairs and CQAs

A general framework for priority-based consistent query answering is presented in [SCM09], where an axiomatic approach is used to specify the desirable properties of preferred repairs that are used to compute CQAs. Denial constraints are considered and (preferred) repairs are maximal consistent subsets of the original database. Preferences among repairs are established by exploiting a priority relation that is an acyclic relation among tuples in the database. It is assumed that the priority relation can be obtained from the user's specifications.

The desirable properties that preferred repairs should satisfy are:

1. the set of preferred repairs cannot be empty;
2. extending the preference relation can only narrow the set of preferred repairs;
3. the set of preferred repairs coincide with the set of repairs if no preference is provided;
4. if the preference establishes a total order among all conflicting tuples then there is a unique preferred repair;
5. every preferred repair is a repair.

Three families of preferred repairs are studied based on different notions of optimality: common optimal (CO) repairs, globally optimal (GO) repairs, and Pareto optimal (PO) repairs.

Functional dependencies are a special kind of denial constraints; the following observation shows the relationship between IMPs and preferred repairs.

Observation 5. *Consider a relation R and a functional dependency fd over R. There exist tuple-based policies γ_{fd} for which there is no CO repair for R w.r.t. fd that is equal to $\gamma_{fd}(R)$; this fact holds even if $\gamma_{fd}(R)$ is consistent w.r.t. fd.*

As an example of such kind of policy, consider a tuple-based policy that, for some cluster, deletes all the tuples in the cluster. As CO repairs are maximal consistent subset of the original database, the result of applying this policy cannot be equal to a CO repair. As every CO repair is a GO repair, that in turn is a PO repair, the above observation holds for GO and PO repairs as well.

Interestingly, PO repairs can be expressed by tuple-based policies.

Proposition 3. *Consider a relation R, a functional dependency fd over R, and a priority relation $\succ$. For every PO repair ρ for R w.r.t. fd and $\succ$, there is a tuple-based policy γ_{fd} such that $\rho = \gamma_{fd}(R)$.*

Proof. PO repairs can be constructed by Algorithm 3 on Page 30 in [SCM09]. This algorithm, constructs a repair R' for R, if R' is Pareto optimal w.r.t. $\succ$ the algorithm finishes returning R'; otherwise, it returns any common optimal repair w.r.t. $\succ$ (it is always possible to compute a common optimal repair in polynomial size of the database and the priority relation, see Algorithm 4 in [SCM09]). Clearly, this algorithm can be seen as a tuple-based IMP, since it satisfies Axioms A1 through A4 and complies with Definition 5. □

Finally, it is easy to see that value- and interval-based IMPs are not expressible in the framework of [SCM09] since attribute-value modifications are not allowed.

3.7 Extensions of Classical Relational Algebra Operators with Multi-Dependency Policies

In this section, we study the relationship between IMPs and classical relational algebra operators. We expand the relational algebra with operators that combine classical relational operators with IMPs. Each relational operator can be augmented by an IMP by either applying the IMP first and then applying the operator, or the other way around. The following definition formalizes this.

Definition 12. *Given two relations R_1 and R_2, a binary relational operator op, two functional dependencies fd$_1$,fd$_2$ defined over R_1 and R_2, respectively, and two IMPs γ_{fd_1} and γ_{fd_2}:*
- A policy-first inconsistency management operator *(*policy-first *operator) is defined as the 7-ary operator $\omega_{\textit{policy-first}}(R_1, R_2, op, fd_1, fd_2, \gamma_{fd_1}, \gamma_{fd_2}) = op(\gamma_{fd_1}(R_1), \gamma_{fd_2}(R_2))$;*
- A policy-last inconsistency management operator *(*policy-last *operator) is defined as the 7-ary operator $\omega_{\textit{policy-last}}(R_1, R_2, op, fd_1, fd_2, \gamma_{fd_1}, \gamma_{fd_2}) = \gamma_{\{fd_1\}\cup\{fd_2\}}(op(R_1, R_2))$.*

Observe that the use of a policy-last operator requires the union of the given sets of functional dependencies to be handled by a multi-dependency policy that takes both sets into account.

In the rest of this section, we study these *policy-first* and *policy-last* operators for several relational algebra operators. The equivalence theorems in this section could form the basis for query optimization in inconsistent DBs (to this aim, appropriate cost models for our policy-first and policy-last operations can be developed). We conclude the section with a discussion about the relationships between IMPs and aggregate functions.

Policy-first vs. *Policy-last* Operators for Projection

In the case of a *policy-last* projection operator, the projection is applied first and then the policy. In order for the inconsistency management operation to make sense, the projected attributes must include all attributes from the left-hand side of the functional dependencies involved in the multi-dependency policy, and at least one of the attributes from the right-hand side of each dependency. This is necessary to ensure that there will still be enough data to be able to identify the inconsistent tuples regarding all functional dependencies involved in the policy.

Example 7. *Consider relation Emp in Example 1 and the functional dependency fd $:$ $Name \rightarrow Salary$. Suppose also that we have a tuple-based policy τ_{fd} for fd that states that in case of inconsistency the tuple with the highest salary must be preferred. Let us consider a projection $\pi_{salary}(Emp)$. If τ_{fd} is applied first to Emp then we obtain $\{80K, 90K\}$. Otherwise, if $\pi_{salary}(Emp)$ is applied first, we no longer have inconsistency w.r.t. fd (and therefore no clusters at all), thus the application of τ_{fd} has no effect and the result is $\{70K, 80K, 90K\}$.*

The situation in this example can arise in the presence of both value- and interval-based IMPs. The following theorem provides necessary and sufficient conditions for which it does not matter if a policy is applied before or after the projection operator.

Theorem 8. *Let R be a relation and fd $: A_1, \ldots, A_k \rightarrow B$ be a functional dependency over R. Let $X \subseteq Attr(R)$ be a superset of $\{A_1, \ldots, A_k, B\}$. For each (tuple-based, value-based, or interval-based) IMP γ_{fd}, $\gamma_{fd}(\pi_X(R)) = \pi_X(\gamma_{fd}(R))$ iff*

1. $\gamma_{fd}(\pi^m_X(R)) = \pi^m_X(\gamma_{fd}(R))$*, and*

2. $\gamma_{fd}(\pi^m_X(R)) = \gamma_{fd}(\pi_X(R))$*,*

where π^m *is the multi-set projection operator (*i.e.*, the projection operator that returns multi-sets rather than sets).*

Proof. ($\Leftarrow$) From conditions (1) and (2) we obtain that $\gamma_{fd}(\pi_X(R)) = \pi^m_X(\gamma_{fd}(R))$. Clearly the first term of this equality is a set, since the application of the policy γ_{fd} on the relation $\pi_X(R)$ is a relation. Hence, $\pi^m_X(\gamma_{fd}(R))$ is also a set. This implies that it is equal to $\pi_X(\gamma_{fd}(R))$.
($\Rightarrow$) Assuming that either condition (1) or (2) is false, it is easy to see that $\gamma_{fd}(\pi_X(R)) \neq \pi_X(\gamma_{fd}(R))$. □

The first condition in Theorem 8 requires that the policy does not depend on the values of attributes not involved in the functional dependency with which the policy is associated. This property is valid for all the IMPs C1-C6 of the Introduction and, intuitively, for a large class of IMPs. The latter condition requires that the policy does not depend on duplicate values of an attribute. This property is valid for IMPs C1, C2, C5, and C6 and for a large class of IMPs like those stating *choose the values lower/greater than (or equal to) a constant*, or *choose all values except the lowest/highest ones*, and so on. The following example shows what can happen when a policy depends on duplicate values.

Example 8. *Consider relation* Emp *of Example 1 and the functional dependency fd* : $Name \rightarrow Salary$. *Let* γ_{fd} *be the policy C3, and* X *be the set* $\{Name, Salary\}$. *Thus,* $\gamma_{fd}(\pi_X(Emp)) = \{(John, 75K), (Mary, 90K)\}$*, but if we apply the policy first, then we have* $\pi_X(\gamma_{fd}(Emp)) = \{(John, 73.33K), (Mary, 90K)\}$. *None of these sets is subset of the other one.*

Policy-first **vs.** ***Policy-last*** **Operators for Selection**

We start by showing that for unrestricted policies, the order in which the policy and the selection operators are applied makes a difference.

Example 9. *Consider the following relation R and the functional dependency fd* : $Name \rightarrow Salary$.

	Name	Salary
t_1	John	70K
t_2	John	80K
t_3	John	90K

Let γ_{fd} be the tuple-based policy "choose all values except the lowest one". *It is easy to see that $\sigma_{Salary \geq 80K}(\gamma_{fd}(R)) = \{t_2, t_3\}$, whereas $\gamma_{fd}(\sigma_{Salary \geq 80K}(R)) = \{t_3\}$. Now assume that the policy is* "choose the lowest value". *We have $\sigma_{Salary \geq 80K}(\gamma_{fd}(R)) = \emptyset$, whereas $\gamma_{fd}(\sigma_{Salary \geq 80K}(R)) = \{t_2\}$.*

Thus, in general, neither $\sigma_C(\gamma_{fd}(R)) \not\subseteq \gamma_{fd}(\sigma_C(R))$ nor $\sigma_C(\gamma_{fd}(R)) \not\supseteq \gamma_{fd}(\sigma_C(R))$. Moreover, as a consequence of Observation 1, this is valid also for the value-based and interval-based approaches. The following proposition identifies a kind of tuple-based policy for which the order is irrelevant.

Proposition 4. *Let R be a relation and fd be a functional dependency over R. Given a tuple-based IMP γ_{fd} and a selection condition C, if there exists a selection condition C_γ such that γ_{fd} is equivalent to σ_{C_γ}, then $\gamma_{fd}(\sigma_C(R)) = \sigma_C(\gamma_{fd}(R))$.*

As an example, suppose we consider the policy γ^1_{fd} for the above example relation. Let γ^1_{fd} be the tuple-based policy *"choose the tuples whose value for Salary is greater than or equal to 85K"*. In this case we have $\sigma_{Salary \geq 80K}(\gamma^1_{fd}(R)) = \{t_3\} = \gamma^1_{fd}(\sigma_{Salary \geq 80K}(R))$. It is easy to see that in this case γ^1_{fd} encodes a condition that can be expressed as a selection condition in the relational algebra.

Policy-first vs. *Policy-last* Operators for Cartesian Product

In this section we discuss the use of the *policy-first* and *policy-last* strategies with the cartesian product operator. The following definition identifies a class of (tuple-based, value-based, or interval-based) IMPs which yield the same result when applied to a multi-set S of tuples or on a multi-set which contains the same proportion of tuples w.r.t. S.

Definition 13. *An IMP γ is said to be* ratio-invariant *iff for any multi-set $S = \{t_1, t_2 \dots, t_n\}$ of tuples and for any integer $k > 0$ it is the case that $\gamma(S) = \gamma(S')$, where S' is the multi-set $S' = \{t_1^i, t_2^i \dots, t_n^i \mid t_j^i = t_j$ where $t_j \in S$ and $i \in [1..k]\}$*

Observe that all IMPs C1-C6 described in the Introduction are ratio-invariant. For instance, policy C1 is ratio-invariant because the minimum value of a multi-set is independent of the number of elements having the same value. The same happens with the weighted mean used by C4: the weighted mean of the values in $\{v_1, v_2 \dots, v_n\}$ is the same of that of $\{v_1^i, v_2^i \dots, v_n^i \mid v_j^i = v_j$ where $v_j \in S$ and $i \in [1..k]\}$ (if $w_j^i = w_j$). On the other hand, a policy stating that *"if there are at least K occurrences of a value V, then choose V, otherwise choose 0"* is not ratio-invariant.

The following theorem provides results regarding the use of (ratio-invariant) IMPs with the Cartesian product operator. Observe that we are assuming the schemas of R_1 and R_2 do not intersect, thus the schema of $R_1 \times R_2$ is the union of the schemas of R_1 and R_2. Moreover, with a little abuse of notation we allow the application of γ_{fd_1} (resp. γ_{fd_2}) to $R_1 \times R_2$, even if they are defined for R_1 (resp. R_2).

Theorem 9. *Let R_1 and R_2 be relations, and fd$_1$ and fd$_2$ be functional dependencies over R_1 and R_2, respectively. For any pair of ratio-invariant (tuple-based, value-based, or interval-based) IMPs γ_{fd_1} and γ_{fd_2}, it is the case that*

1. $\gamma_{fd_1}(R_1 \times R_2) = \gamma_{fd_1}(R_1) \times R_2$*;*

2. $\gamma_{fd_2}(R_1 \times R_2) = R_1 \times \gamma_{fd_2}(R_2)$*;*

3. $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \times R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$*;*

4. $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \times R_2)) = \gamma_{fd_2}(\gamma_{fd_1}(R_1 \times R_2))$*.*

Proof. We now prove the first case, the second case can be proved by analogous reasoning. Since fd_1 works only on the attributes of R_1 and is ratio-invariant, it holds that $\gamma_{fd_1}(\pi^m_{Attr(R_1)}(R_1 \times R_2)) = \gamma_{fd_1}(R_1^M) = \gamma_{fd_1}(R_1)$, where π^m is the multi-set projection operator, and R_1^M is the multi-set relation obtained by copying $M = |R_2|$ times every tuple of R_1. As $R_1 \times R_2 = \{t_1.t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2\}$, it is easy that $\gamma_{fd_1}(R_1 \times R_2)$ is equivalent to the set $\{t'_1.t_2 \mid t'_1 \in \gamma_{fd_1}(R_1) \wedge t_2 \in R_2\}$, which is $\gamma_{fd_1}(R_1) \times R_2$.

As to the third case, since $\gamma_{fd_2}(R_1 \times R_2) = R_1 \times \gamma_{fd_2}(R_2)$, as a consequence of the first statement we obtain $\gamma_{fd_1}(R_1 \times \gamma_{fd_2}(R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$.

The result of the fourth case follows from the fact that $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \times R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$ and $\gamma_{fd_2}(\gamma_{fd_1}(R_1 \times R_2)) = \gamma_{fd_1}(R_1) \times \gamma_{fd_2}(R_2)$. □

Policy-first vs. *Policy-last* Operators for θ-Join

We now discuss the classical θ-join operator [Ull88], which is defined as $R_1 \bowtie_\theta R_2 = \sigma_\theta(R_1 \times R_2)$ where R_1 and R_2 are relations and θ is a predicate over *Attr*$(R_1) \cup$ *Attr*(R_2). As we have already studied policy-first and policy-last operators for selection and Cartesian product and this operator is defined as a simple combination of the two, we have the following result.

Corollary 1. *Let* R_1 *and* R_2 *be relations, and* fd_1 *and* fd_2 *be functional dependencies over* R_1 *and* R_2*, respectively. For any pair of ratio-invariant tuple-based IMPs* γ_{fd_1} *and* γ_{fd_2}*, if* γ_{fd_1} *and* γ_{fd_2} *are equivalent to* σ_{C_1} *and* σ_{C_2}*, respectively, for some selection conditions* C_1 *and* C_2*, then:*

1. $\gamma_{fd_1}(R_1 \bowtie_\theta R_2) = \gamma_{fd_1}(R_1) \bowtie_\theta R_2$*;*

2. $\gamma_{fd_2}(R_1 \bowtie_\theta R_2) = R_1 \bowtie_\theta \gamma_{fd_2}(R_2)$*;*

3. $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \bowtie_\theta R_2)) = \gamma_{fd_1}(R_1) \bowtie_\theta \gamma_{fd_2}(R_2)$*;*

4. $\gamma_{fd_1}(\gamma_{fd_2}(R_1 \bowtie_\theta R_2)) = \gamma_{fd_2}(\gamma_{fd_1}(R_1 \bowtie_\theta R_2))$.

Policy-first vs. *Policy-last* Operators for Union

In this section we discuss the use of the *policy-first* and *policy-last* strategies with the union operator. We consider the case when the union is performed between two relations having the same schema and the functional dependency is defined over this schema. We start by showing a case where different results are obtained when the policy is applied before or after the union.

Example 10. *Consider the relational schema* $S(\mathit{Name}, \mathit{Salary})$ *and the functional dependency fd* $: Name \rightarrow Salary$*. Assume that relation* R_1 *is* $\{t_1 = (John, 75K),$ $t_2 = (John, 80K)\}$ *and* R_2 *is* $\{t_3 = (John, 70K), t_4 = (John, 85K), t_5 = (Mary, 90K)\}$*. Let* γ_{fd} *be the tuple-based policy* “choose the tuples with highest value of Salary”*. It is easy to see that* $\gamma_{fd}(R_1) = \{t_2\}$*,* $\gamma_{fd}(R_2) = \{t_4, t_5\}$*, and* $\gamma_{fd}(R_1 \cup R_2) = \{t_4, t_5\}$*. Hence,* $\gamma_{fd}(R_1 \cup R_2) \not\supseteq \gamma_{fd}(R_1) \cup \gamma_{fd}(R_2)$*. Now assume that the policy is* “choose all the values except the lowest one”*. We obtain* $\gamma_{fd}(R_1) = \{t_2\}$*,* $\gamma_{fd}(R_2) = \{t_4, t_5\}$*, but in this case* $\gamma_{fd}(R_1 \cup R_2) = \{t_1, t_2, t_4, t_5\}$*. Hence,* $\gamma_{fd}(R_1 \cup R_2) \not\subseteq \gamma_{fd}(R_1) \cup \gamma_{fd}(R_2)$*.*

Similar cases can be identified for value-based and interval-based policies. The following theorem presents a sufficient condition for which it does not matter if a policy is applied before or after the union.

Theorem 10. *Let R_1 and R_2 be relations over the relational schema S, and fd $: X \rightarrow B$ be a functional dependency where $X \subseteq Attr(S)$. For any (tuple-based, value-based, or interval-based) IMPs γ_{fd}, $\gamma_{\mathit{fd}}(R_1 \cup R_2) = \gamma_{\mathit{fd}}(R_1) \cup \gamma_{\mathit{fd}}(R_2)$ if $\pi_X(R_1) \cap \pi_X(R_2) = \emptyset$.*

Proof. If $\pi_X(R_1) \cap \pi_X(R_2) = \emptyset$ then $clusters(R_1, \{fd\}) \cap clusters(R_2, \{fd\}) = \emptyset$, and $clusters(R_1 \cup R_2, \{fd\}) = clusters(R_1, \{fd\}) \cup clusters(R_2, \{fd\})$. Thus, the application of γ_{fd} to $R_1 \cup R_2$ yields the same result as when it is applied first on the portion of $R_1 \cup R_2$ corresponding with R_1 and then on the portion corresponding to R_2. □

Policy-first vs. *Policy-last* Operators for Difference

We discuss the use of the *policy-first* and *policy-last* strategies with the difference operator.

Theorem 11. *Let R_1 and R_2 be relations over schema S and fd be a functional dependency over S. For any tuple-based IMP γ_{fd}, it is the case that: $\gamma_{\mathit{fd}}(R_1 - R_2) \subseteq R_1 - \gamma_{\mathit{fd}}(R_2)$.*

Proof. Suppose that $\gamma_{\mathit{fd}}(R') \nsubseteq R_1 - \gamma_{\mathit{fd}}(R_2)$. If this is the case, let t be a tuple in $\gamma_{\mathit{fd}}(R')$ such that $t \notin R_1 - \gamma_{\mathit{fd}}(R_2)$. This means that either (1) $t \notin R_1$ or (2) $t \in R_1$ and $t \in \gamma_{\mathit{fd}}(R_2)$. Let us analyze each case separately:

- If $t \notin R_1$ then $t \notin R' = R_1 - R_2$, therefore $t \notin \gamma_{\mathit{fd}}(R')$ since γ_{fd} is a tuple-based policy, which is a contradiction.

- If $t \in R_1$ and $t \in \gamma_{\mathit{fd}}(R_2)$ then, given that γ_{fd} is a tuple-based policy, it must be the case that $t \in R_2$. However, this means that $t \notin R' = R_1 - R_2$ and $t \notin \gamma_{\mathit{fd}}(R')$, which yields a contradiction.

□

Moreover, the following example shows that $\gamma_{fd}(R_1) - R_2 \not\subseteq \gamma_{fd}(R_1 - R_2)$.

Example 11. *Consider the relations* $R_1 = \{(a,-1),(a,1),(a,2)\}$ *and* $R_2 = \{(a,-1)\}$ *over the schema* $S(A,B)$ *with* $fd : A \to B$*, and the tuple-based policy stating that for each cluster, if there are no negative values then remove everything, otherwise do nothing. Then,* $\gamma_{fd}(R_1) = R_1$ *and* $\gamma_{fd}(R_1) - R_2 = \{(a,1),(a,2)\}$*. However,* $\gamma_{fd}(R_1 - R_2) = \gamma_{fd}(\{(a,1),(a,2)\}) = \emptyset$ *which is not a superset of* $\gamma_{fd}(R_1) - R_2$*.*

Relationship with Aggregate Functions

We conclude by discussing the relationships of the *policy-first* strategies with the most common aggregate functions: count, max, min, sum, and avg. With a little abuse of notation, we consider a relational algebraic expression of the form

$$G\, \mathfrak{F}_{\mathsf{f}(A)}(R)$$

where f is an aggregate function, equivalent to an SQL query of the form

```
SELECT G, f(A) AS A′ FROM R GROUP BY G
```

Moreover, we write $G\, \mathfrak{F}_{\mathsf{f}(A)}(R_1) \sqsubseteq G\, \mathfrak{F}_{\mathsf{f}(A)}(R_2)$ to denote that $\forall(t_1,t_2)$ such that $t_1 \in G\, \mathfrak{F}_{\mathsf{f}(A)}(R_1)$, $t_2 \in G\, \mathfrak{F}_{\mathsf{f}(A)}(R_2)$, $t_1[G] = t_2[G]$, it holds that $t_1[A'] \leq t_2[A']$.

Example 12. *Consider relation* Emp *of Example 1. The relational algebraic expression* $Name\, \mathfrak{F}_{\mathsf{AVG}(Salary)}(Emp)$ *corresponds to an SQL query of the form*

```
SELECT Name, AVG(Salary) AS Salary′ FROM Emp GROUP BY Name.
```

The result of the expression is the relation $\{(John, 73.33K), (Mary, 90K)\}$ *with attributes* $Name$ *and* $Salary'$*. Now consider a relation*

Emp$' = \{(John, 100K, 30), (John, 110K, 30), (Mary, 90K, 30), (Mary, 100K, 25)\}$.

The result of the application of the above expression to Emp′ *is the relation* $\{(John, 105K), (Mary, 95K)\}$*. We can write*

$Name\ \mathfrak{F}_{\textsf{AVG}(Salary)}(Emp) \sqsubseteq Name\ \mathfrak{F}_{\textsf{AVG}(Salary)}(Emp')$

since the value of $Salary'$ *for both John and Mary is lower in the relation on the left-hand side.*

The above notation is trivially extended to the $\sqsupseteq$ and $=$ operators.

Proposition 5. *Let* R *be a relation over schema* S*, fd a functional dependency over* R*, and* E *an attribute in* $S \setminus (LHS(fd) \cup RHS(fd))$*. The following relationships hold:*

- *For any IMP* γ_{fd}*,*
  - $\mathfrak{F}_{count(*)}(\gamma_{fd}(R)) \sqsubseteq \mathfrak{F}_{count(*)}(R)$;
  - $LHS(fd)\mathfrak{F}_{count(*)}(\gamma_{fd}(R)) \sqsubseteq LHS(fd)\mathfrak{F}_{count(*)}(R)$.
- *For any IMP* γ_{fd}*,*
  - $\mathfrak{F}_{max(RHS(fd))}(\gamma_{fd}(R)) \sqsubseteq \mathfrak{F}_{max(RHS(fd))}(R)$;
  - $LHS(fd)\mathfrak{F}_{max(RHS(fd))}(\gamma_{fd}(R)) \sqsubseteq LHS(fd)\mathfrak{F}_{max(RHS(fd))}(R)$.
- *For any tuple-based IMP* γ_{fd}*,*
  - $\mathfrak{F}_{max(E)}(\gamma_{fd}(R)) \sqsubseteq \mathfrak{F}_{max(E)}(R)$;
  - $LHS(fd)\mathfrak{F}_{max(E)}(\gamma_{fd}(R)) \sqsubseteq LHS(fd)\mathfrak{F}_{max(E)}(R)$.
- *For any value- or interval-based IMP* γ_{fd}*,*
  - $\mathfrak{F}_{max(E)}(\gamma_{fd}(R)) = \mathfrak{F}_{max(E)}(R)$;
  - $LHS(fd)\mathfrak{F}_{max(E)}(\gamma_{fd}(R)) = LHS(fd)\mathfrak{F}_{max(E)}(R)$.

- *For any IMP γ_{fd},*
  - $\mathfrak{F}_{min(RHS(fd))}(\gamma_{fd}(R)) \sqsupseteq \mathfrak{F}_{min(RHS(fd))}(R)$;
  - $LHS(fd)\mathfrak{F}_{min(RHS(fd))}(\gamma_{fd}(R)) \sqsupseteq LHS(fd)\mathfrak{F}_{min(RHS(fd))}(R)$.
- *For any tuple-based IMP γ_{fd},*
  - $\mathfrak{F}_{min(E)}(\gamma_{fd}(R)) \sqsupseteq \mathfrak{F}_{min(E)}(R)$;
  - $LHS(fd)\mathfrak{F}_{min(E)}(\gamma_{fd}(R)) \sqsupseteq LHS(fd)\mathfrak{F}_{min(E)}(R)$.
- *For any value- or interval-based IMP γ_{fd},*
  - $\mathfrak{F}_{min(E)}(\gamma_{fd}(R)) = \mathfrak{F}_{min(E)}(R)$;
  - $LHS(fd)\mathfrak{F}_{min(E)}(\gamma_{fd}(R)) = LHS(fd)\mathfrak{F}_{min(E)}(R)$.
- *Let γ_{fd} be the interval-based IMP that replaces all inconsistent values with the corresponding cluster's average. Then,*

  $LHS(fd)\mathfrak{F}_{avg(RHS(fd))}(\gamma_{fd}(R)) = LHS(fd)\mathfrak{F}_{avg(RHS(fd))}(R)$.
- *Let γ_{fd} be the value-based IMP that replaces all inconsistent values with their minimum. Then:*
  - $\mathfrak{F}_{sum(RHS(fd))}(\gamma_{fd}(R)) \sqsubseteq \mathfrak{F}_{sum(RHS(fd))}(R)$;
  - $\mathfrak{F}_{min(RHS(fd))}(\gamma_{fd}(R)) = \mathfrak{F}_{min(RHS(fd))}(R)$;
  - $\mathfrak{F}_{avg(RHS(fd))}(\gamma_{fd}(R)) \sqsubseteq \mathfrak{F}_{avg(RHS(fd))}(R)$;
  - $LHS(fd)\mathfrak{F}_{sum(RHS(fd))}(\gamma_{fd}(R)) \sqsubseteq LHS(fd)\mathfrak{F}_{sum(RHS(fd))}(R)$;
  - $LHS(fd)\mathfrak{F}_{min(RHS(fd))}(\gamma_{fd}(R)) = LHS(fd)\mathfrak{F}_{min(RHS(fd))}(R)$;
  - $LHS(fd)\mathfrak{F}_{avg(RHS(fd))}(\gamma_{fd}(R)) \sqsubseteq LHS(fd)\mathfrak{F}_{avg(RHS(fd))}(R)$.

- *Let γ_{fd} be the value-based IMP that replaces all inconsistent values with their maximum. Then:*
    - $\mathfrak{F}_{\textsf{sum}(RHS(fd))}(\gamma_{fd}(R)) \sqsupseteq \mathfrak{F}_{\textsf{sum}(RHS(fd))}(R)$;
    - $\mathfrak{F}_{\textsf{max}(RHS(fd))}(\gamma_{fd}(R)) = \mathfrak{F}_{\textsf{max}(RHS(fd))}(R)$;
    - $\mathfrak{F}_{\textsf{avg}(RHS(fd))}(\gamma_{fd}(R)) \sqsupseteq \mathfrak{F}_{\textsf{avg}(RHS(fd))}(R)$;
    - $LHS(fd)\mathfrak{F}_{\textsf{sum}(RHS(fd))}(\gamma_{fd}(R)) \sqsupseteq LHS(fd)\mathfrak{F}_{\textsf{sum}(RHS(fd))}(R)$;
    - $LHS(fd)\mathfrak{F}_{\textsf{max}(RHS(fd))}(\gamma_{fd}(R)) = LHS(fd)\mathfrak{F}_{\textsf{max}(RHS(fd))}(R)$;
    - $LHS(fd)\mathfrak{F}_{\textsf{avg}(RHS(fd))}(\gamma_{fd}(R)) \sqsupseteq LHS(fd)\mathfrak{F}_{\textsf{avg}(RHS(fd))}(R)$.

3.8 Applying IMPs

In this section, we tackle the problem: *how can we implement IMPs efficiently*? The question of implementing inconsistency management approaches efficiently has not been addressed to date because most past works try to address very general KBs. Furthermore, even when simple kinds of KBs are used, efforts such as those proposed by the consistent query answering community are intractable [CLR03].

The heart of the problem of applying an IMP lies in the fact that the clusters must be identified. Thus, we start by discussing how classical DBMS indexes can be used to carry out these operations, and then we present a new data structure that can be used to identify the set of clusters more efficiently: the *cluster table*.

3.8.1 Using DBMS-based Indexes

A basic approach to the problem of identifying clusters is to directly define one DBMS index (DBMSs in general provide hash indexes, B-trees, etc.) for each functional dependency's left-hand side. Assuming that the DBMS index used allows $O(1)$ access to individual tuples, this approach has several advantages:

- Takes advantage of the highly optimized implementation of operations which is provided by the DBMS. Insertion, deletion, lookup, and update are therefore all inexpensive operations in this case.

- Identifying a single cluster (for given values for the left-hand side of a certain functional dependency) can be done by issuing a simple query to the DBMS, which can be executed in $O(\max_{cl \in clusters(R,fd)} |cl|)$ time, in the (optimistic) assumption of $O(1)$ time for accessing a single tuple. However, the exact cost depends on the particular DBMS implementation, especially that of the query planner.

- Identifying all clusters can be done in two steps, each in time in $O(|R|)$:

  1. issue a query with a GROUP BY on the left-hand side of the functional dependency of interest and count the number of tuples associated with each one;

  2. take those left-hand side values with a count greater than one and obtain the cluster.

  This can be easily done in a single nested query.

There is, however, one important disadvantage to this approach: clusters must be identified time and time again, and are not explicitly maintained. This means that, in situations where a large portion of the table constitutes clean tuples (and we therefore have few clusters), the $O(|R|)$ operations associated with obtaining all clusters become quite costly because they may entail actually going through the entire table.

3.8.2 Cluster Table

We now introduce a data structure that we call *cluster table*. For each $fd \in \mathcal{F}$, we maintain a cluster table focused on that one dependency. When relation R gets updated, each FD's associated cluster table must be updated. This section defines the cluster table associated with an FD, how that cluster table gets updated, and how it can be employed to efficiently apply an IMP. Note that even though we do not cover the application of multiple policies, we assume that for each relation a set of cluster tables associated with $\mathcal{F}$ must be maintained. Therefore, when a policy w.r.t. an FD is applied to a relation, the cluster tables corresponding to other FDs in $\mathcal{F}$ might need to be updated as well. Moreover, we make the assumption that the application of a policy can be done on a cluster-by-cluster basis, *i.e.*, applying a policy to a relation has the same effect as applying the policy to every cluster independently. This is a rather important class of policies because (*i*) they are intuitive from the user viewpoint, as they are easy to specify, and it is also easy to reason about the effects they will have on the relations they are applied on; (*ii*) all repairing strategies for functional dependency violations in the database research literature work in this manner; (*iii*) they are easy to enforce in a policy specification language.

Definition 14 (Tuple group). *Given a relation R and a set of attributes $\mathcal{A} \subseteq$ Attr(R), a* tuple group w.r.t. *$\mathcal{A}$ is a maximal set $g \subseteq R$ such that $\forall t, t' \in g, t[\mathcal{A}] = t'[\mathcal{A}]$.*

We use *groups*$(R, \mathcal{A})$ to denote the set of all tuple groups in R w.r.t. $\mathcal{A}$, and M to denote the maximum size of a group, *i.e.*, $M = \max_{g \in \mathit{groups}(R,\mathcal{A})} |g|$. The following result shows that all clusters are groups, but not vice-versa.

Proposition 6. *Given a relation R and a functional dependency fd defined over R, clusters(R, fd) $\subseteq$ groups$(R, LHS(fd))$.*

The reason a group may not be a cluster is because the FD may be satisfied by the tuples in the group. In the cluster table approach, we store all groups associated with a table together with an indication of whether the group is a cluster or not. When tuples are inserted into the relation, or when they are deleted or modified, the cluster table can be easily updated using procedures we will present shortly.

Definition 15 (Cluster table)**.** *Given a relation R and a functional dependency fd, a* cluster table *w.r.t.* (R,fd)*, denoted* $ct(R,fd)$ *is a pair* (G, D) *where:*

- *G is a set containing, for each tuple group $g \in groups(R, LHS(fd))$ s.t. $|g| > 1$, a tuple of the form $(v, \overrightarrow{g}, flag)$, where:*
  - *$v = t[LHS(fd)]$ where $t \in g$;*
  - *$\overrightarrow{g}$ is a set of pointers to the tuples in g;*
  - *$flag$ is $true$ iff $g \in$ clusters(R,fd), $false$ otherwise.*
- *D is a set of pointers to the tuples in $R \setminus \bigcup_{g \in groups(R,LHS(fd)),|g|>1} g$;*
- *both G and D are sorted by $LHS(fd)$.*

Example 13. *Consider the* Flight *relation in Fig. 3.2. This relation has the schema $Flight(Aline, FNo, Orig, Dest, Deptime, Arrtime)$ where $dom(Aline)$ is a finite set of airline codes, $dom(FNo)$ is the set of all flight numbers, $dom(Orig)$ and $dom(Dest)$ are the airport codes of all airports in the world, and $dom(Deptime)$ and $dom(Arrtime)$ is the set of all times expressed in military time (*e.g.*, 1425 hrs or 1700 hours and so forth).*[5] *In this case, $fd = Aline, FNo \rightarrow Orig$ might be an FD that says that each (Aline,FNo) pair uniquely determines an origin.*

[5] For the sake of simplicity, we are not considering cases where flights arrive on the day after departure, etc. – these can be accommodated through an expanded schema.

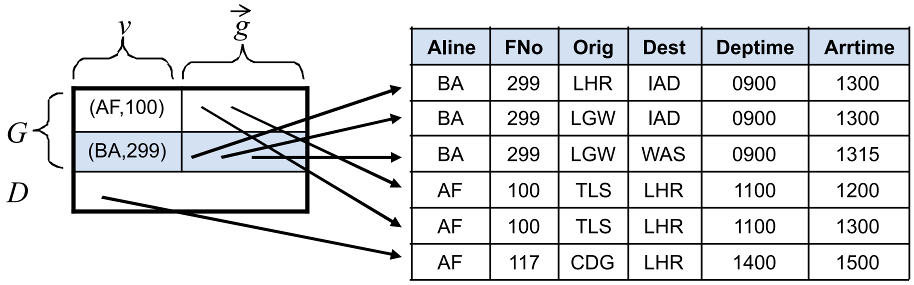
	Aline	FNo	Orig	Dest	Deptime	Arrtime	*Source*
t_1	BA	299	LHR	IAD	0900	1300	s_1
t_2	BA	299	LGW	IAD	0900	1300	s_2
t_3	BA	299	LGW	WAS	0900	1315	s_3
t_4	AF	100	TLS	LHR	1100	1200	s_1
t_5	AF	100	TLS	LHR	1100	1300	s_5
t_6	AF	117	CDG	LHR	1400	1500	s_6

Figure 3.2: Example relation

It is easy to see that $\{t_1, t_2\}$ *and* $\{t_1, t_3\}$ *are culprits w.r.t.* $(Flight, \{fd\})$*, and the only cluster is* $\{t_1, t_2, t_3\}$*. Moreover,* $\{t_1, t_2, t_3\}$ *is a group in groups*$(Flight, \{Aline, FNo\})$*, as are* $\{t_4, t_5\}$ *and* $\{t_6\}$ *– but* $\{t_4, t_5\}$ *and* $\{t_6\}$ *are not clusters. For this relation, the cluster table* $ct(Flight,fd)$ *has the following form:*

$$G = \{((AF, 100), \{\overrightarrow{t_4}, \overrightarrow{t_5}\}, false), ((BA, 299), \{\overrightarrow{t_1}, \overrightarrow{t_2}, \overrightarrow{t_3}\}, true)\},\ D = \{\overrightarrow{t_6}\}.$$

A graphical representation of $ct(Flight,fd)$ *is shown in Fig. 3.3.*



Aline	FNo	Orig	Dest	Deptime	Arrtime
BA	299	LHR	IAD	0900	1300
BA	299	LGW	IAD	0900	1300
BA	299	LGW	WAS	0900	1315
AF	100	TLS	LHR	1100	1200
AF	100	TLS	LHR	1100	1300
AF	117	CDG	LHR	1400	1500

Figure 3.3: Cluster table $ct(Flight,fd)$. Shaded rows in G correspond to groups flagged as clusters

A cluster table can be built through a simple procedure that, given an FD *fd*, identifies the clusters w.r.t. *fd* in a relation R by first sorting the tuples in R according to the left-hand side of *fd*, then performing a linear scan of the ordered list of tuples.

Proposition 7. *Given a relation* R *and a functional dependency fd defined over* R*, the worst-case running time for building* $ct(R,fd)$ *is* $O(|R| \cdot log|R|)$*.*

Maintaining cluster tables

We now study how to update a cluster table for a relation R and a set *fd* under three kinds of updates: (*i*) when a tuple is inserted into R, (*ii*) when a tuple is deleted from R, and (*iii*) when a tuple in R is modified.

Insertion. Fig. 3.4 describes an algorithm for updating $ct(R{,}fd)$ after inserting a new tuple t in R. The algorithm starts by checking whether t belongs to a tuple group already present in R (line 1) and, if this is the case, it (*i*) adds $\overrightarrow{t}$ to the corresponding entry in G and (*ii*) checks if the group is a cluster (lines 3–6). If t does not already belong to a tuple group, the algorithm checks whether it forms a new group when paired to a tuple pointed to by D (line 8). If this is the case, it adds the new group to G (lines 9–11); otherwise, it just adds $\overrightarrow{t}$ to D (line 13).

```
    Algorithm CT-insert
    Input: Relation $R$, functional dependency fd, cluster
           table $(G, D) = ct(R, fd)$, new tuple $t$

 1  if $\exists(t[LHS(fd)], \overrightarrow{g}, flag) \in G$ then
 2    add $\overrightarrow{t}$ to $\overrightarrow{g}$
 3    if $flag = false$ then
 4      pick the first $\overrightarrow{t'}$ from $\overrightarrow{g}$
 5      if $t[RHS(fd)] \neq t'[RHS(fd)]$ then
 6        $flag \leftarrow true$
 7    end-algorithm
 8  if $\exists \overrightarrow{t'} \in D$ s.t. $t[LHS(fd)] = t'[LHS(fd)]$ then
 9    remove $\overrightarrow{t'}$ from $D$
10    add $(t[LHS(fd)], \{\overrightarrow{t}, \overrightarrow{t'}\}, flag)$ to $G$ where
11          $flag = true$ iff $t[RHS(fd)] \neq t'[RHS(fd)]$
12    end-algorithm
13  add $\overrightarrow{t}$ to $D$
```

Figure 3.4: Updating a cluster table after inserting a tuple

The following example shows how this algorithm works.

Example 14. *Consider the cluster tables for Example 13 and suppose a new tuple* $t = (AF, 100, CDG, LHR, 1100, 1200)$ *is inserted into relation* Flight. *Algorithm CT-insert* *first adds $\overrightarrow{t}$ to set $\overrightarrow{g}$ in the first row of the cluster table. Then, it picks $\overrightarrow{t_4}$ and, since $t[RHS(fd)] \neq t_4[RHS(fd)]$, it assigns $true$ to the $flag$ of the first row. Now suppose that the new tuple is $t = (AF, 117, CDG, LHR, 1400, 1500)$. In this case, the algorithm removes $\overrightarrow{t_6}$ from D and, since $t[RHS(fd)] = t_6[RHS(fd)]$, adds a new row $((AF, 117), \{\overrightarrow{t}, \overrightarrow{t_6}\}, false)$ to G.*

The following results ensure the correctness and complexity of CT-insert.

Proposition 8. *Algorithm CT-insert terminates and correctly computes $ct(R \cup \{t\}, fd)$.*

Proposition 9. *The worst-case running time of Algorithm CT-insert is $O(\log(|G|) + \log(|D|))$.*

Deletion. Fig. 3.5 presents an algorithm for updating a cluster table $ct(R, fd)$ after deleting a tuple t from R. The algorithm checks whether t belongs to a tuple group (line 1) and, if this is the case, it removes $\overrightarrow{t}$ from the corresponding entry in G and checks if the group is a cluster (lines 3–4). Otherwise, it just removes $\overrightarrow{t}$ from D (line 6).

```
   Algorithm CT-delete
   Input: Relation $R$, functional dependency fd, cluster
          table $(G, D) = ct(R, fd)$, deleted tuple $t$
1  if $\exists(t[LHS(fd)], \overrightarrow{g}, flag) \in G$ then
2    remove $\overrightarrow{t}$ from $\overrightarrow{g}$
3    if $|\overrightarrow{g}| = 1$ then
4      remove $(t[LHS(fd)], \overrightarrow{g}, flag)$ from $G$
5      add $\overrightarrow{g}$ to $D$
6    else if $flag = true$ and $\nexists \overrightarrow{t_1}, \overrightarrow{t_2} \in \overrightarrow{g}$ s.t.
           $t_1[RHS(fd)] \neq t_2[RHS(fd)]$ then
7      $flag \leftarrow false$
8    end-algorithm
9  remove $\overrightarrow{t}$ from $D$
```

Figure 3.5: Updating a cluster table after deleting a tuple

Example 15. *Consider the cluster tables for Example 13 and suppose tuple t_5 is removed from relation* Flight. *Algorithm CT-delete first removes $\overrightarrow{t_5}$ from set $\overrightarrow{g}$ in the first row of* *the cluster table. Then, since the group has been reduced to a singleton, it moves $\overrightarrow{t}$ to set D and removes the first row from G. Now suppose that tuple t_1 is removed from the relation* Flight. *In this case, the algorithm removes $\overrightarrow{t_1}$ from set $\overrightarrow{g}$ in the second row of the cluster table. As the group is no longer a cluster (t_2 and t_3 agree on the $Orig$ attribute), the algorithm sets the corresponding $flag$ to $false$.*

The following results specify the correctness and complexity of the CT-delete algorithm.

Proposition 10. *Algorithm CT-delete terminates and correctly computes $ct(R \setminus \{t\}, fd)$.*

Proposition 11. *The worst-case running time of Algorithm CT-delete is* $O(\log(|G|) + \log(|D|) + M)$.

Update. Fig. 3.6 shows an algorithm for updating a cluster table $ct(R, fd)$ after updating a tuple t to t' in R (clearly, $\overrightarrow{t} = \overrightarrow{t'}$).

```
    Algorithm CT-update
    Input: Relation $R$, functional dependency fd, cluster
           table $(G, D) = ct(R, fd)$, tuples $t, t'$

1   if $t[LHS(fd)] = t'[LHS(fd)]$ and $t[RHS(fd)] = t'[RHS(fd)]$ then
2     end-algorithm
3   if $t[LHS(fd)] = t'[LHS(fd)]$ and $\exists(t[LHS(fd)], \overrightarrow{g}, flag) \in G$ then
4     if $flag = true$ and $\nexists \overrightarrow{t''} \in \overrightarrow{g}$ s.t. $t'[RHS(fd)] \neq t''[RHS(fd)]$ then
5       $flag \leftarrow false$
6       end-algorithm
7     if $flag = false$ then
8       pick the first $\overrightarrow{t''}$ from $\overrightarrow{g}$
9       if $t'[RHS(fd)] \neq t''[RHS(fd)]$ then
10        $flag \leftarrow true$
11    end-algorithm
12  if $t[LHS(fd)] = t'[LHS(fd)]$ then
13    end-algorithm
14  execute CT-delete with $t$
15  execute CT-insert with $t'$
```

Figure 3.6: Updating a cluster table after updating a tuple

The algorithm first checks whether anything regarding *fd* has changed in the update (lines 1–2). If this is the case and t belongs to a group (line 3), the algorithm checks if the group was a cluster whose inconsistency has been removed by the update (lines 4–6) or the other way around (lines 7-11). At this point, as t does not belong to any group and the values of the attributes in the left-hand side of *fd* did not change, the algorithm ends (lines 12–13) because this means that the updated tuple simply remains in D. If none of the above conditions apply, the algorithm simply calls CT-delete and then CT-insert.

Example 16. *Consider the cluster table for the flight example (Example 13). Suppose the value of the $Orig$ attribute of tuple t_1 is changed to LGW. Tuple t_1 belongs to the group represented by the second row in G, which is a cluster. However, after the update to t, no two tuples in the group have different values of $Orig$, and thus Algorithm CT-update changes the corresponding $flag$ to $false$. Now suppose the value of the $Orig$ attribute of tuple t_5 is changed to LGW. In this case, the algorithm picks $\overrightarrow{t_4}$ and, since $t_4[RHS(\mathit{fd})] \neq t_5[RHS(\mathit{fd})]$, it assigns $true$ to the $flag$ of the second row.*

The following results specify the correctness and complexity of CT-update.

Proposition 12. *Algorithm CT-update terminates and correctly computes $ct(R \setminus \{t\} \cup \{t'\},\mathit{fd})$.*

Proposition 13. *The worst-case running time of Algorithm CT-update is $O(\log(|G|) + \log(|D|) + M)$.*

Applying cluster tables to compute IMPs

We now show how to use the cluster table to compute an IMP over an RKB with an FD. Fig. 3.7 shows the proposed algorithm. For each cluster in G, procedure $apply(\gamma_{\mathit{fd}}, \overrightarrow{g})$ applies policy γ_{fd} to the set of tuples $\overrightarrow{g}$. As a consequence of this, depending on the nature of the policy, some tuples in $\overrightarrow{g}$ might be deleted or updated according to what the policy determines. Therefore, $changes$ keeps the list on changes performed by the policy in a cluster. After applying the IMP, the algorithm updates the cluster table in order to preserve its integrity; it checks whether all the inconsistencies have been removed (lines 3–4) and whether the cluster has been reduced to a single tuple (lines 5–7). The first check is necessary so the flag can be updated and future applications of a policy do not need to consider that group if it is no longer a cluster; in the latter case, the pointer is moved from G to D since that tuple is no longer in conflict with any other tuple w.r.t. fd. Finally, the changes performed by the policy are propagated to the rest of the cluster tables for relation R. That is, for every other functional dependency in $\mathcal{F}$ either *CT-delete* or *CT-update* are called on the corresponding cluster table depending on the nature of the change.

```
Algorithm CT-applyIMP
Input: Relation $R$, functional dependency fd, cluster table $(G, D) = ct(R, \mathit{fd})$, IMP $\gamma_{\mathit{fd}}$

1   for all $(v, \overrightarrow{g}, true) \in G$
2     $changes \leftarrow apply(\gamma_{\mathit{fd}}, \overrightarrow{g})$
3     if $\nexists \overrightarrow{t_1}, \overrightarrow{t_2} \in \overrightarrow{g}$ s.t. $t_1[RHS(\mathit{fd})] \neq t_2[RHS(\mathit{fd})]$ then
4       $flag \leftarrow false$
5     if $|\overrightarrow{g}| = 1$ then
6       remove $(v, \overrightarrow{g}, true)$ from $G$
7       add $\overrightarrow{g}$ to $D$
8     for all $\mathit{fd}' \in \mathcal{F}$ and $\mathit{fd}' \neq \mathit{fd}$
9       let $(G', D')$ be the cluster table associated with $\mathit{fd}'$
10      for all change $ch \in changes$
11        if $ch = delete(t, R)$ then CT-delete$(R, \mathit{fd}', (G', D'), t)$
12        if $ch = update(t, t', R)$ then CT-update$(R, \mathit{fd}', (G', D'), t, t')$
```

Figure 3.7: Applying an IMP using a cluster table

The following results show the correctness and complexity of the CT-applyIMP algorithm.

Proposition 14. *Algorithm CT-applyIMP terminates and correctly computes* $\gamma_{\mathit{fd}}(R)$ *and* $ct(\gamma_{\mathit{fd}}(R), \mathit{fd})$.

Proposition 15. *The worst-case time complexity of CT-applyIMP is* $O(|G| \cdot (poly(M) + log|D| + |\mathcal{F}| \cdot |M| \cdot (log|G'| + log|D'| + M')))$*, where* G' *(resp.* D'*) is the largest set* G *(resp.* D*) among all cluster tables, and* M' *is the maximum* M *among all cluster tables.*

In the next section we will present the results of our experimental evaluation of cluster tables vs. the DBMS-based approach discussed above.

3.8.3 Experimental Evaluation

Our experiments measure the running time performance of applying IMPs using cluster tables; moreover, we analyzed the required storage space on disk. We compared these measures with those obtained through the use of a heavily optimized DBMS-based index. The parameters varied were the size of the database and the amount of inconsistency present.

Our prototype JAVA implementation consists of roughly 9,000 lines of code, relying on Berkeley DB Java Edition[6] database for implementation of our disk-based index structures. The DBMS-based index was implemented on top of PostgreSQL version 7.4.16; a B-Tree index (PostgreSQL does not currently allow hash indexes on more than one attribute) was defined for the LHS of each functional dependency. All experiments were run on multiple multi-core Intel Xeon E5345 processors at 2.33GHz, 8GB of memory, running the Scientific Linux distribution of the GNU/Linux operating system (our implementation makes use of only 1 processor and 1 core at a time, the cluster is used for multiple runs). The numbers reported are the result of averaging between 5 and 50 runs to minimize experimental error. All tables had 15 attributes and 5 functional dependencies associated with them. Tables were randomly generated with a certain percentage of inconsistent tuples[7] divided in clusters of 5 tuples each. The cluster tables were implemented on top of BerkeleyDB; for each table, both G and D were kept in the hash structures provided by BerkeleyDB.

[6] *http://www.oracle.com/database/berkeley-db/je/index.html*

[7] Though of course tuples themselves are not inconsistent, we use this term to refer to tuples that are involved in some inconsistency, *i.e.*, belong to a cluster.

Fig. 3.8 shows comparisons of policy application times when varying the size of the database and the percentage of inconsistent tuples. The operation carried out was the application of a value-based policy that replaces the right-hand side of tuples in a cluster with the median value in the cluster; this policy was applied to all clusters in the table.

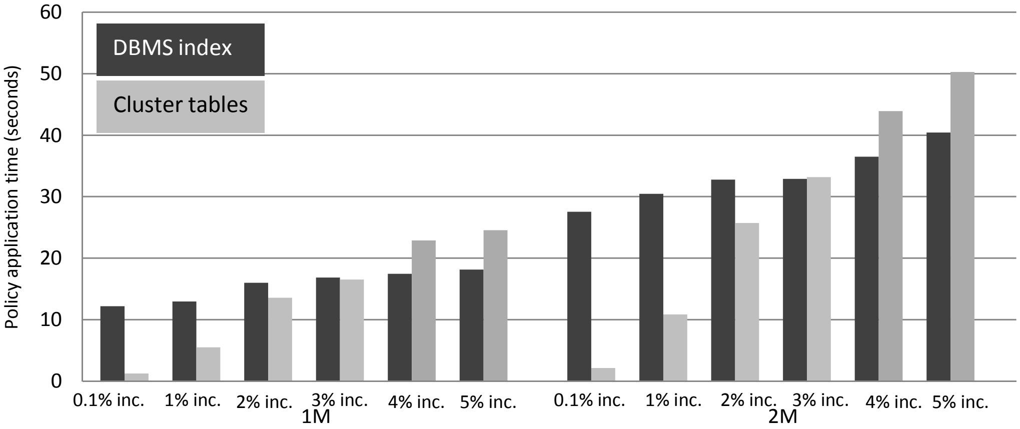


Figure 3.8: Average policy application times for (*i*) 1M and 2M tuples and (*ii*) varying percentage of inconsistency

We can see that the amount of inconsistency clearly affected the cluster table-based approach more than it did the DBMS-based index. For the runs with less than 3% inconsistent tuples, the cluster table outperformed the DBMS-based approach (in particular, in the case of a database with 2 million tuples and 0.1% inconsistency, applying the policy took 2.12 seconds with the cluster table and 27.56 seconds with the DBMS index). This is due to the fact that relatively few clusters are present and thus many tuples can be ignored, while the DBMS index must process all of them. Overall, our experiments suggest that under about 3% inconsistency the cluster table approach is able to provide much better performance in the application of IMPs. Further experiments with 0.1% inconsistency (Fig. 3.9) show that the cluster table approach remains quite scalable over much larger databases, while the performance of the DBMS index degrades quickly – for a database with 5 millon tuples, applying the policy took 3.7 seconds with the cluster table and 82.9 seconds with the DBMS index.

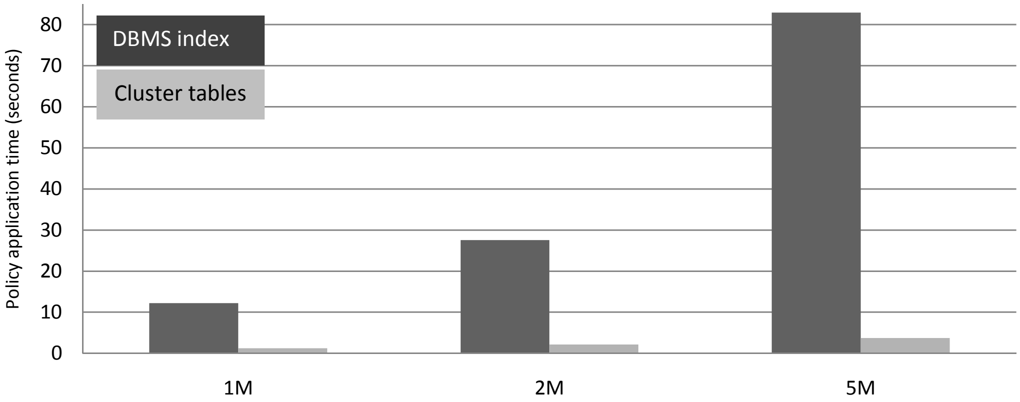


Figure 3.9: Average policy application times for 0.1 percentage of inconsistency for tables from 1M to 5M tuples

Fig. 3.10 shows comparisons of disk footprints when varying the size of the database and the percentage of inconsistent tuples – note that the numbers reported include the sizes of the structures needed to index all of the functional dependencies used in the experiments.

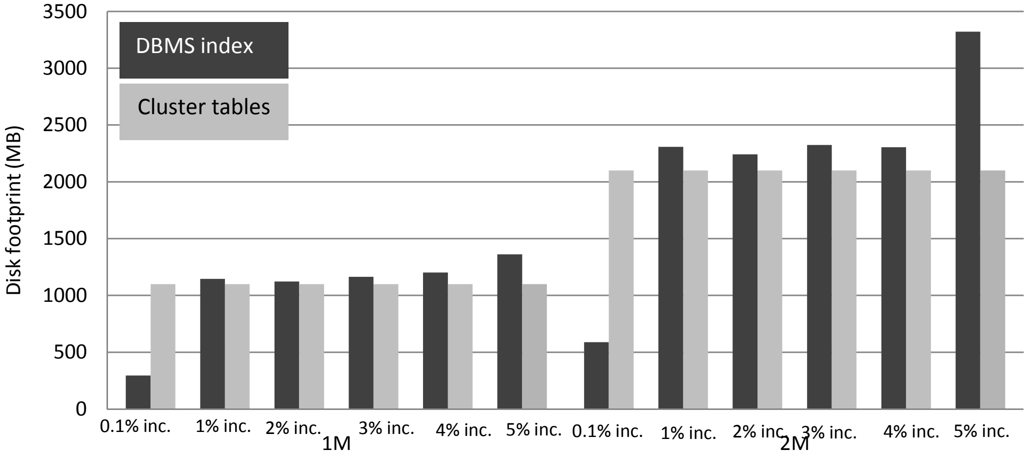


Figure 3.10: Disk footprint for (*i*) 1M and 2M tuples and (*ii*) varying percentage of inconsistency

In this case, the cluster table approach provides a smaller footprint with respect to the DBMS index in all cases except when 0.1% inconsistency is present. In the case of a database with 2 million tuples and 5% inconsistency, the cluster tables size was 63% of that of the DBMS index.[8]

In performing update operations, the cluster table approach performed at most 1 order of magnitude worse than the DBMS index. This result is not surprising since these kinds of operations are the specific target of DBMS indexes, which are thus able to provide extremely good performance in these cases (*e.g.*, 2 seconds for 1,000 update operations over a database containing 2 million tuples).

Overall, our evaluation showed that the cluster table approach is capable of providing very good performance in scenarios where 1%-3% inconsistency is present, which are extremely common [BFFR05]. For lower inconsistency, the rationale behind this approach becomes even more relevant and makes the application of IMPs much faster.

3.9 Concluding Remarks

None of the past approaches to inconsistency management is capable of handling cases C3, C4, C5, and C6 presented in the Introduction because past approaches adhere to three important tenets: first, that no "new" data should be introduced into the database; second, that as much of the original data as possible should be retained, and third, that consistency must be restored.

Though we agree these are sometimes desirable goals, the fact remains that users in specific application domains often know a lot more about the intricacies of *their* data than a database designer who has never seen the data. In many of these cases, the end-user wants to resolve inconsistencies by taking (*i*) *his knowledge of the data into account* (which the DB designer has no chance of knowing a priori) and (*ii*) his mission risk into

[8] In addition, we point out that our current implementation is not yet optimized for an intelligent use of disk space, as the DBMS is.

account — which also the DB designer has no chance of knowing a priori. Tools for managing inconsistent data today do not support such users.

For example, there are many cases where end-users might actually want to introduce “seemingly new” data – in case C3 and C4, the user wants to take an average or weighted average of salaries. This may be what the user or his company determines is appropriate for his application domain. Should he be stopped from doing this by database designers who do not know the application *a priori*? No.

Likewise, consider case C6. When conducting a scientific experiment (biological, atmospheric, etc.), inconsistent data might be collected for any number of reasons (faulty measurements, incorrectly mixed chemicals, environmental factors). Should the results of the experiments be based on *dirty* data? Some scientists at least would argue “No” (perhaps for some experiments) and eliminate the dirty data and repeat all, or parts, of the experiment. Databases should provide support for decisions users want to make, not make decisions for them that users don’t like.

In response to these needs, we introduced in this work the concept of *inconsistency management policies* as functions satisfying a *minimal* set of axioms. We proposed several IMPs families that satisfy these axioms, and study relations between them in the simplified case where only one functional dependency is present. We show that when multiple FDs are present, multiple alternative semantics can result. We introduced new versions of the relational algebra that are augmented by inconsistency management policies that are applied either before the operator or after. We develop theoretical results on the resulting extended relational operators that could, in principle, be used in the future for query optimization. Furthermore, we propose different approaches for implementing an IMP-based framework and show that it is versatile, can be implemented based on the needs and resources of the user and, according to our theoretical and experimental results, the associated algorithms incur reasonable costs. As a consequence, IMPs are a powerful tool for end users to express what they wish to do with their data, rather than have a system manager or a DB engine that does not understand their domain problem to dictate how they should handle inconsistencies in their data.

Chapter 4

A General Framework for Reasoning about Inconsistency

The work presented in this chapter is taken from [MM$^+$11].

4.1 Introduction

Inconsistency management, as reviewed in Chapter 1 has been intensely studied in various parts of AI, often in slightly disguised form [Gar88a, PL92, Poo85, RM70]. All the excellent works described in 1 provide an *a priori* conflict resolution mechanism. A user who uses a system based on these papers is forced to use the semantics implemented in the system, and has little say in the matter (besides which most users querying KBs are unlikely to be experts in even classical logic, let alone default logics and argumentation methods).

The aims of this chapter are:

1. to propose a unified framework for reasoning about inconsistency, which captures existing approaches as a special case and provides an easy basis for comparison;

2. to apply the framework using any monotonic logic, including ones for which inconsistency management has not been studied before (*e.g.*, temporal, spatial, and probabilistic logics), and provide new results on the complexity of reasoning about inconsistency in such logics;

3. to allow end-users to bring their domain knowledge to bear, allowing them to voice an opinion on *what works for them, not what a system manager decided was right for them*, in other words, to take into account the preferences of the end-user;

4. to propose the concept of an *option* that specifies the semantics of an inconsistent theory in any of these monotonic logics and the notion of a preferred option that takes the user's domain knowledge into account; and

5. to propose general algorithms for computing the preferred options.

We do this by building upon Alfred Tarski and Dana Scott's celebrated notion of an abstract logic. We start with a simple example to illustrate why conflicts can often end up being resolved in different ways by human beings, and why it is important to allow end-users to bring their knowledge to bear when a system resolves conflicts. A database system designer or an AI knowledge base designer cannot claim to understand *a priori* the specifics of each application that his knowledge base system may be used for in the future.

Example 17. *Suppose a university payroll system says that John's salary is 50K, while the university personnel database says it is 60K. In addition, there may be an axiom that says that everyone has exactly one salary. One simple way to model this is via the theory*

S below.

$$salary(John, 50K) \leftarrow \tag{4.1}$$

$$salary(John, 60K) \leftarrow \tag{4.2}$$

$$S_1 = S_2 \leftarrow salary(X, S_1) \wedge salary(X, S_2). \tag{4.3}$$

The above theory is obviously inconsistent. Suppose (4.3) is definitely known to be true. Then, a bank manager considering John for a loan may choose the 50K number to determine the maximum loan amount that John qualifies for. On the other hand, a national tax agency may use the 60K figure to send John a letter asking him why he underpaid his taxes.

Neither the bank manager nor the tax officer is making any attempt to find out the truth (thus far); however, both of them are making different decisions based on the same facts.

The following examples present theories expressed in different logics which are inconsistent – thus the reasoning that can be done is very limited. We will continue these examples later on to show how the proposed framework is suitable for handling all these scenarios in a flexible way.

Example 18. *Consider the temporal logic theory T below. The $\bigcirc$ operator denotes the "next time instant" operator. Thus, the first rule says that if* received *is true at time t (intuitively, a request is received at time t), then* processed *is true at time $t+1$ (the* *request is processed at the next time point).*

$$\bigcirc processed \leftarrow received. \tag{4.4}$$

$$received. \tag{4.5}$$

$$\bigcirc \neg processed. \tag{4.6}$$

Clearly, the theory is inconsistent. Nevertheless, there might be several options that a user might consider to handle it: for instance, one might replace the first rule with a "weaker" one stating that if a request is received, it will be processed sometime in the future, not necessarily at the next time point.

Example 19. *Consider the probabilistic logic theory P consisting of three formulas $p : [0.3, 0.4]$, $p : [0.41, 0.43]$, $p : [0.44, 0.6]$. In general, the formula $p : [\ell, u]$ says that the probability of proposition p lies in the interval $[\ell, u]$. The theory is easily seen to be inconsistent under this informal reading. A user might choose to resolve the inconsistency in different ways, such as by discarding a minimal set of formulas or modifying the probability intervals as little as possible.*

The rest of this chapter proceeds as follows. In Section 4.2, we recall Tarski's notion of what an abstract logic is [Tar56]. Then, in Section 4.3, we define our general framework for reasoning about inconsistency for any Tarskian logic. In Section 4.4, we develop general algorithms to compute preferred options based on various types of assumptions. In Section 4.5, we show applications of our framework to several monotonic logics for which no methods to reason about inconsistency exist to date - these logics include probabilistic logics [Hal90, Nil86a], linear temporal logic of the type used extensively in model checking [Eme90, GPSS80], fuzzy logic, Levesque's logic of belief [Lev84] and spatial logic captured via the region connection calculus [RCC92]. In many of these cases, we are able to establish new results on the complexity of reasoning about inconsistency in such logics. In Section 4.6, we show how certain existing works can be captured in our general framework.

4.2 Tarski's Abstract Logic

Alfred Tarski [Tar56] defines an *abstract logic* as a pair $(\mathcal{L}, \mathsf{CN})$ where the members of $\mathcal{L}$ are called *well-formed formulas*, and CN is a *consequence operator*. CN is any function from $2^{\mathcal{L}}$ (the powerset of $\mathcal{L}$) to $2^{\mathcal{L}}$ that satisfies the following axioms (here X is a subset of $\mathcal{L}$):

1. $X \subseteq \mathsf{CN}(X)$ **(Expansion)**

2. $\mathsf{CN}(\mathsf{CN}(X)) = \mathsf{CN}(X)$ **(Idempotence)**

3. $\mathsf{CN}(X) = \bigcup_{Y \subseteq_f X} \mathsf{CN}(Y)$ **(Finiteness)**

4. $\mathsf{CN}(\{x\}) = \mathcal{L}$ for some $x \in \mathcal{L}$ **(Absurdity)**

Notation: $Y \subseteq_f X$ means that Y is a finite subset of X.

Intuitively, $\mathsf{CN}(X)$ returns the set of formulas that are logical consequences of X according to the logic in question. It can be easily shown from the above axioms that CN is a closure operator, that is, for any $X, X', X'' \subseteq \mathcal{L}$, CN enjoys the following properties:

5. $X \subseteq X' \Rightarrow \mathsf{CN}(X) \subseteq \mathsf{CN}(X')$. **(Monotonicity)**

6. $\mathsf{CN}(X) \cup \mathsf{CN}(X') \subseteq \mathsf{CN}(X \cup X')$.

7. $\mathsf{CN}(X) = \mathsf{CN}(X') \Rightarrow \mathsf{CN}(X \cup X'') = \mathsf{CN}(X' \cup X'')$.

Almost all well-known monotonic logics (such as propositional logic [Sho67], first order logic [Sho67], modal logic, temporal logic, fuzzy logic, probabilistic logic [FHM90], etc.) can be viewed as special cases of Tarski's notion of an abstract logic. AI introduced non-monotonic logics [Bob80] which do not satisfy the monotonicity property.

Once $(\mathcal{L}, \mathsf{CN})$ is fixed, a notion of *consistency* arises as follows.

Definition 16 (Consistency)**.** *Let* $X \subseteq \mathcal{L}$*.* X *is* consistent *w.r.t. the logic* $(\mathcal{L}, \mathsf{CN})$ *iff* $\mathsf{CN}(X) \neq \mathcal{L}$*. It is* inconsistent *otherwise.*

The previous definition says that X is consistent iff its set of consequences is not the set of all formulas. For any abstract logic $(\mathcal{L}, \mathsf{CN})$, we also require the following axiom to be satisfied:

8. $\mathsf{CN}(\emptyset) \neq \mathcal{L}$ **(Coherence)**

The coherence requirement (absent from Tarski's original proposal, but added here to avoid considering trivial systems) forces the empty set $\emptyset$ to always be consistent - this makes sense for any reasonable logic as saying emptiness should intuitively be consistent.

It can be easily verified that if a set $X \subseteq \mathcal{L}$ is consistent, then its closure under CN is consistent as well as any subset of X. Moreover, if X is inconsistent, then every superset of X is inconsistent.

4.3 A General Framework for Handling Inconsistency

This section proposes a general framework for handling inconsistency under any monotonic logic. Basically, our approach to reason with an inconsistent knowledge base (KB) is a process which follows three steps:

1. Determining consistent "subbases";

2. Selecting among all the subbases the ones that are *preferred*;

3. Applying entailment on the preferred subbases.

Throughout the rest of this chapter, we assume that we have an arbitrary, but fixed *monotonic* logic $(\mathcal{L}, \mathsf{CN})$.

The basic idea behind our framework is to construct what we call *options*, and then to define a preference relation on these options. The *preferred options* are intended to support the conclusions to be drawn from the inconsistent knowledge base. Intuitively, an option is a set of formulas that is both consistent and closed w.r.t. consequence in logic $(\mathcal{L}, \mathsf{CN})$.

Definition 17 (Options)**.** *An* option *is any set $\mathcal{O}$ of elements of $\mathcal{L}$ such that:*

- *$\mathcal{O}$ is consistent.*

- *$\mathcal{O}$ is closed,* i.e., *$\mathcal{O} = \mathsf{CN}(\mathcal{O})$.*

We use $\mathtt{Opt}(\mathcal{L})$ to denote the set of all options that can be built from $(\mathcal{L}, \mathsf{CN})$.

Note that the empty set is not necessarily an option. This depends on the value of $\mathsf{CN}(\emptyset)$ in the considered logic $(\mathcal{L}, \mathsf{CN})$. For instance, in propositional logic, it is clear that $\emptyset$ is not an option since all the tautologies will be inferred from it. Indeed, it is easy to see that $\emptyset$ is an option iff $\mathsf{CN}(\emptyset) = \emptyset$.

Clearly, for each consistent subset X of $\mathcal{L}$, it holds that $\mathsf{CN}(X)$ is an option (as $\mathsf{CN}(X)$ is consistent and Idempotence axiom entails that $\mathsf{CN}(X)$ is closed). Since we are considering generic logic, we can show that options do not always exist.

Proposition 16. *The set $\mathtt{Opt}(\mathcal{L}) = \emptyset$ iff*

1. $\forall \psi \in \mathcal{L}$, $\mathsf{CN}(\{\psi\})$ is inconsistent, and

2. $\mathsf{CN}(\emptyset) \neq \emptyset$.

Proof. ($\Rightarrow$) Let us assume that $\mathtt{Opt}(\mathcal{L})$ = $\emptyset$. Let us also assume by contradiction that $\exists\psi \in \mathcal{L}$ such that $\mathsf{CN}(\{\psi\})$ is consistent. Since $\mathsf{CN}(\{\psi\})$ is closed by the Idempotence axiom, $\mathsf{CN}(\{\psi\})$ is an option, which is a contradiction.

Assume now that $\mathsf{CN}(\emptyset)$ = $\emptyset$. This means that $\emptyset$ is an option since it is closed and consistent ($\mathsf{CN}(\emptyset) \neq \mathcal{L}$), which is a contradiction.

($\Leftarrow$) Let us assume that i) $\forall\psi \in \mathcal{L}$, $\mathsf{CN}(\{\psi\})$ is inconsistent, and ii) $\mathsf{CN}(\emptyset) \neq \emptyset$. Assume also by contradiction that $\mathtt{Opt}(\mathcal{L}) \neq \emptyset$ and let $\mathcal{O} \in \mathtt{Opt}(\mathcal{L})$. There are two cases:

Case 1: $\mathcal{O} = \emptyset$. Consequently, $\mathsf{CN}(\emptyset)$ = $\emptyset$, which contradicts assumption ii).

Case 2: $\mathcal{O} \neq \emptyset$. Since $\mathcal{O}$ is consistent, $\exists\psi \in \mathcal{O}$ s.t. $\{\psi\}$ is consistent and thus $\mathsf{CN}(\{\psi\})$ is consistent. This contradicts assumption i).

□

So far, we have defined the concept of option for any logic $(\mathcal{L}, \mathsf{CN})$ in a way that is independent of a knowledge base. We now show how to associate a set of options with an inconsistent knowledge base.

In most approaches for handling inconsistency, the maximal consistent subsets of a given inconsistent knowledge base have an important role. This may induce one to think of determining the options of a knowledge base as the closure of its maximal consistent subsets. However, this approach has the side effect of dropping entire formulas, whereas more fine-grained approaches could be adopted in order to preserve more information of the original knowledge base. This is shown in the following example.

Example 20. *Consider the propositional knowledge base* $\mathcal{K} = \{(a \wedge b); \neg b\}$. *There are two maximal consistent subsets, namely* $MCS_1 = \{a \wedge b\}$ *and* $MCS_2 = \{\neg b\}$. *However,* *one could argue that MCS_2 is too weak, since we could have included a by "weakening" the formula $(a \wedge b)$ instead of dropping it altogether.*

The "maximal consistent subset" approach, as well as the one suggested in the previous example, can be seen as a particular case of a more general approach, where one considers consistent "relaxations" (or *weakenings*) of a given inconsistent knowledge base. The ways in which such weakenings are determined might be different, as the following examples show.

Example 21. *Consider again the temporal knowledge base of Example 18. An intuitive way to "weaken" the knowledge base might consist of replacing the $\bigcirc$ (*next moment in time*) connective with the $\Diamond$ (*sometime in the future*) connective. So, for instance,* $\bigcirc$processed $\leftarrow$ received *might be replaced by* $\Diamond$processed $\leftarrow$ received*, thus saying that if* received *is true at time t, then* processed *is true at some subsequent time $t' \geq t$ (not necessarily at time $t+1$). This would lead to a consistent knowledge base, whose closure is clearly an option. Likewise, we might weaken only (4.6), obtaining another consistent knowledge base whose closure is an option.*

Example 22. *Consider the probabilistic knowledge base of Example 19. A reasonable way to make a probabilistic formula $\phi : [\ell, u]$ weaker, might be to replace it with another formula $\phi : [\ell', u']$ where $[\ell, u] \subseteq [\ell', u']$.*

The preceding examples suggest that a flexible way to determine the options of a given knowledge base should be provided, since what is considered reasonable to be an option might depend on the logic and the application domain at hand, and, more importantly, it should depend on the user's preferences. The basic idea is to consider *weakenings* of a given knowledge base $\mathcal{K}$ whose closures yield options. For instance, as said before, weakenings might be subsets of the knowledge base. Although such a weakening mechanism is general enough to be applicable to many logics, more tailored mechanisms could be defined for specific logics. For instance, the two reasonable approaches illustrated in Example 21 and 22 above cannot be captured by considering subsets of the original knowledge bases; as another example, let us reconsider Example 20: by looking at subsets of the knowledge base, it is not possible to get an option containing both a and $\neg b$. We formally introduce the notion of $weakening$ as follows.

Definition 18. *Given an element ψ of $\mathcal{L}$,*

$$weakening(\psi) = \begin{cases} \mathsf{CN}(\{\psi\}) & if\ \psi\ is\ consistent \\ \emptyset & otherwise \end{cases}$$

Definition 19. *Given a knowledge base $\mathcal{K}$,*

$$weakening(\mathcal{K}) = \{\mathcal{K}' \mid \forall \psi' \in \mathcal{K}'\ (\exists \psi \in \mathcal{K}.\ \psi' \in weakening(\psi))\}$$

According to the preceding definitions, to weaken a knowledge base intuitively means to weaken formulas in it; to weaken a formula ψ means to take some formulas in $\mathsf{CN}(\{\psi\})$ if ψ is consistent, or to otherwise drop ψ altogether (note that a consistent formula could also be dropped). $weakening(\mathcal{K})$ can be computed by first finding $weakening(\psi)$ for all $\psi \in \mathcal{K}$ and then returning the subsets of $\bigcup_{\psi \in \mathcal{K}} weakening(\psi)$. It is easy to see that if $\mathcal{K}' \in weakening(\mathcal{K})$, then $\mathsf{CN}(\mathcal{K}') \subseteq \mathsf{CN}(\mathcal{K})$.

Observe that although a knowledge base in $weakening(\mathcal{K})$ does not contain any inconsistent formulas, it could be inconsistent.

Definition 20. *A* weakening mechanism *is a function $\mathcal{W} : 2^{\mathcal{L}} \rightarrow 2^{2^{\mathcal{L}}}$ such that $\mathcal{W}(\mathcal{K}) \subseteq$ $weakening(\mathcal{K})$ for any $\mathcal{K} \in 2^{\mathcal{L}}$.*

The preceding definition says that a weakening mechanism is a function that maps a knowledge base into knowledge bases that are *weaker* in some sense. For instance, an example of a weakening mechanism is $\mathcal{W}(\mathcal{K}) = weakening(\mathcal{K})$. This returns *all* the weaker knowledge bases associated with $\mathcal{K}$. We use $\mathcal{W}_{all}$ to denote this weakening mechanism.

We now define the set of options for a given knowledge base (w.r.t. a selected weakening mechanism).

Definition 21. *Let $\mathcal{K}$ be a knowledge base in logic $(\mathcal{L}, \mathsf{CN})$ and $\mathcal{W}$ a weakening mechanism. We say that an option $\mathcal{O} \in \mathtt{Opt}(\mathcal{L})$ is an option for $\mathcal{K}$ (w.r.t. $\mathcal{W}$) iff there exists $\mathcal{K}'$ in $\mathcal{W}(\mathcal{K})$ such that $\mathcal{O} = \mathsf{CN}(\mathcal{K}')$.*

Thus, an option for $\mathcal{K}$ is the closure of some weakening $\mathcal{K}'$ of $\mathcal{K}$. Clearly, $\mathcal{K}'$ must be consistent because $\mathcal{O}$ is consistent (by virtue of being an option) and because $\mathcal{O} = \mathsf{CN}(\mathcal{K}')$. In other words, the options for $\mathcal{K}$ are the closure of consistent weakenings of $\mathcal{K}$. We use $\mathtt{Opt}(\mathcal{K}, \mathcal{W})$ to denote the set of options for $\mathcal{K}$ under the weakening mechanism $\mathcal{W}$. Whenever $\mathcal{W}$ is clear from the context, we simply write $\mathtt{Opt}(\mathcal{K})$ instead of $\mathtt{Opt}(\mathcal{K}, \mathcal{W})$.

Note that if we restrict $\mathcal{W}(\mathcal{K})$ to be $\{\mathcal{K}' \mid \mathcal{K}' \subseteq \mathcal{K}\}$, Definition 21 corresponds to that presented in [SA07] (we will refer to such a weakening mechanism as $\mathcal{W}_{\subseteq}$). Moreover, observe that every option for a knowledge base w.r.t. this weakening mechanism is also an option for the knowledge base when $\mathcal{W}_{all}$ is adopted, that is, the options obtained in the former case are a subset of those obtained in the latter case.

Example 23. *Consider again the knowledge base of Example 20 and let $\mathcal{W}_{all}$ be the adopted weakening mechanism. Our framework is flexible enough to allow to have the set $\mathsf{CN}(\{a, \neg b\})$ as an option for $\mathcal{K}$. This weakening mechanism preserves more information from the original knowledge base than the classical "maximal consistent subsets" approach.*

In Section 4.5 we will consider specific monotonic logics and show more tailored weakening mechanisms.

The framework for reasoning about inconsistency has three components: the set of all options for a given knowledge base, a preference relation between options, and an inference mechanism.

Definition 22 (General framework). *A* general framework for reasoning about inconsistency *in a knowledge base $\mathcal{K}$ is a triple $\langle \mathtt{Opt}(\mathcal{K},\mathcal{W}), \succeq, \mid\!\sim \rangle$ such that:*

- $\mathtt{Opt}(\mathcal{K},\mathcal{W})$ *is the set of options for $\mathcal{K}$ w.r.t. the weakening mechanism $\mathcal{W}$.*
- $\succeq \subseteq \mathtt{Opt}(\mathcal{K},\mathcal{W}) \times \mathtt{Opt}(\mathcal{K},\mathcal{W})$. $\succeq$ *is a partial (or total) preorder (*i.e.*, it is reflexive and transitive).*
- $\mid\!\sim\ : 2^{\mathtt{Opt}(\mathcal{K},\mathcal{W})} \to \mathtt{Opt}(\mathcal{L})$.

The second important concept of the general framework above is the preference relation $\succeq$ among options. Indeed, $\mathcal{O}_1 \succeq \mathcal{O}_2$ means that the option $\mathcal{O}_1$ is at least as preferred as $\mathcal{O}_2$. This relation captures the idea that some options are better than others because, for instance, the user has decided that this is the case, or because those preferred options satisfy the requirements imposed by the developer of a conflict management system. For instance, in Example 17, the user chooses certain options (*e.g.*, the options where the salary is minimal or where the salary is maximal based on his needs). From the partial preorder $\succeq$ we can derive the strict partial order $\succ$ (*i.e.*, it is irreflexive and transitive) over $\mathtt{Opt}(\mathcal{K},\mathcal{W})$ as follows: for any $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K},\mathcal{W})$ we say $\mathcal{O}_1 \succ \mathcal{O}_2$ iff $\mathcal{O}_1 \succeq \mathcal{O}_2$ and $\mathcal{O}_2 \not\succeq \mathcal{O}_1$. Intuitively, $\mathcal{O}_1 \succ \mathcal{O}_2$ means that $\mathcal{O}_1$ is *strictly* preferable to $\mathcal{O}_2$. The set of *preferred* options in $\mathtt{Opt}(\mathcal{K},\mathcal{W})$ determined by $\succeq$ is $\mathtt{Opt}^{\succeq}(\mathcal{K},\mathcal{W}) = \{\mathcal{O} \mid \mathcal{O} \in \mathtt{Opt}(\mathcal{K},\mathcal{W}) \wedge \nexists \mathcal{O}' \in \mathtt{Opt}(\mathcal{K},\mathcal{W}) \text{ with } \mathcal{O}' \succ \mathcal{O}\}$. Whenever $\mathcal{W}$ is clear from the context, we simply write $\mathtt{Opt}^{\succeq}(\mathcal{K})$ instead of $\mathtt{Opt}^{\succeq}(\mathcal{K},\mathcal{W})$.

In the following three examples, we come back to the example theories of Section 4.1 to show how our framework can handle them.

Example 24. *Let us consider again the knowledge base S of Example 17. Consider the options $\mathcal{O}_1 = \mathsf{CN}(\{(1),(3)\})$, $\mathcal{O}_2 = \mathsf{CN}(\{(1),(2)\})$, $\mathcal{O}_3 = \mathsf{CN}(\{(2),(3)\})$, and let us say that these three options are strictly preferable to all other options for S; then, we have to determine the preferred options among these three. Different criteria might be used to determine the preferred options:*

- *Suppose the* score *$sc(\mathcal{O}_i)$ of option $\mathcal{O}_i$ is the sum of the elements in the multiset $\{S \mid salary(John, S) \in \mathcal{O}_i\}$. In this case, the score of $\mathcal{O}_1$ is $50K$, that of $\mathcal{O}_2$ is $110K$, and that of $\mathcal{O}_3$ is $60K$. We could now say that $\mathcal{O}_i \succeq \mathcal{O}_j$ iff $sc(\mathcal{O}_i) \leq sc(\mathcal{O}_j)$. In this case, the only preferred option is $\mathcal{O}_1$, which corresponds to the bank manager's viewpoint.*

- *On the other hand, suppose we say that $\mathcal{O}_i \succeq \mathcal{O}_j$ iff $sc(\mathcal{O}_i) \geq sc(\mathcal{O}_j)$. In this case, the only preferred option is $\mathcal{O}_2$; this corresponds to the view that the rule saying everyone has only one salary is wrong (perhaps the database has John being paid out of two projects simultaneously and $50K$ of his salary is charged to one project and $60K$ to another).*

- *Now consider the case where we change our scoring method and say that $sc(\mathcal{O}_i) = \mathsf{min}\{S \mid salary(John, S) \in \mathcal{O}_i\}$. In this case, $sc(\mathcal{O}_1) = 50K$, $sc(\mathcal{O}_2) = 50K$, $sc(\mathcal{O}_3) = 60K$. Let us suppose that the preference relation says that $\mathcal{O}_i \succeq \mathcal{O}_j$ iff $sc(\mathcal{O}_i) \geq sc(\mathcal{O}_j)$. Then, the only preferred option is $\mathcal{O}_3$, which corresponds exactly to the tax agency's viewpoint.*

Example 25. *Let us consider the temporal logic theory T of Example 18. We may choose to consider just three options for determining the preferred ones: $\mathcal{O}_1 = \mathsf{CN}(\{(4.4),(4.5)\})$,*

$\mathcal{O}_2 = \mathsf{CN}(\{(4.4), (4.6)\})$, $\mathcal{O}_3 = \mathsf{CN}(\{(4.5), (4.6)\})$. *Suppose now that we can associate a numeric score with each formula in* T*, describing the reliability of the source that provided the formula. Let us say these scores are 3, 1, and 2 for formulas (4.4), (4.5) and (4.6), respectively, and the weight of an option* $\mathcal{O}_i$ *is the sum of the scores of the formulas in* $T \cap \mathcal{O}_i$*. We might say* $\mathcal{O}_i \succeq \mathcal{O}_j$ *iff the score of* $\mathcal{O}_i$ *is greater than or equal to the score of* $\mathcal{O}_j$*. In this case, the only preferred option is* $\mathcal{O}_2$*.*

Example 26. *Consider the probabilistic logic theory* P *of Example 19. Suppose that in order to determine the preferred options, we consider only options that assign a single non-empty probability interval to* p*, namely options of the form* $\mathsf{CN}(\{p : [\ell, u]\})$*. For two atoms* $A_1 = p : [\ell_1, u_1]$ *and* $A_2 = p : [\ell_2, u_2]$*, let diff*$(A_1, A_2) = abs(\ell_1 - \ell_2) + abs(u_1 - u_2)$*. Let us say that the score of an option* $\mathcal{O} = \mathsf{CN}(\{A\})$*, denoted by* $score(\mathcal{O})$*, is given by* $\sum_{A' \in P}$ *diff*(A, A')*. Suppose we say that* $\mathcal{O}_i \succeq \mathcal{O}_j$ *iff* $score(\mathcal{O}_i) \leq score(\mathcal{O}_j)$*. Intuitively, this means that we are preferring options that change the lower and upper bounds in* P *as little as possible. In this case,* $\mathsf{CN}(\{p : [0.41, 0.43]\})$ *is a preferred option.*

Thus, we see that our general framework for managing inconsistency is very powerful - it can be used to handle inconsistencies in different ways based upon how the preference relation between options is defined. In Section 4.5, we will consider more logics and illustrate more examples showing how the proposed framework is suitable for handling inconsistency in a flexible way.

The following definition introduces a preference criterion where an option is preferable to another if and only if the latter is a weakening of the former.

Definition 23. *Consider a knowledge base* $\mathcal{K}$ *and a weakening mechanism* $\mathcal{W}$*. Let* $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K}, \mathcal{W})$*. We say* $\mathcal{O}_1 \succeq_W \mathcal{O}_2$ *iff* $\mathcal{O}_2 \in weakening(\mathcal{O}_1)$*.*

Proposition 17. *Consider a knowledge base $\mathcal{K}$ and a weakening mechanism $\mathcal{W}$. Let $\mathcal{O}_1, \mathcal{O}_2 \in \texttt{Opt}(\mathcal{K}, \mathcal{W})$. $\mathcal{O}_1 \succeq_W \mathcal{O}_2$ iff $\mathcal{O}_1 \supseteq \mathcal{O}_2$.*

Proof. ($\Rightarrow$) Let $\psi_2 \in \mathcal{O}_2$. By definition of $\succeq_W$, there exists $\psi_1 \in \mathcal{O}_1$ s.t. $\psi_2 \in weakening(\psi_1)$; that is $\psi_2 \in \mathsf{CN}(\{\psi_1\})$. Since $\{\psi_1\} \subseteq \mathcal{O}_1$, it follows that $\mathsf{CN}(\{\psi_1\}) \subseteq \mathcal{O}_1$ (by Monotonicity and the fact that $\mathcal{O}_1$ is closed). Hence, $\psi_2 \in \mathcal{O}_1$.

($\Leftarrow$) Let $\psi_2 \in \mathcal{O}_2$. Clearly, $\psi_2 \in weakening(\psi_2)$, since ψ_2 is consistent and $\psi_2 \in \mathsf{CN}(\{\psi_2\})$ (Expansion axiom). As $\psi_2 \in \mathcal{O}_1$, the condition expressed in Definition 19 trivially holds and $\mathcal{O}_1 \succeq_W \mathcal{O}_2$. □

The following corollary states that $\succeq_W$ is indeed a preorder (in particular, a partial order).

Corollary 2. *Consider a knowledge base $\mathcal{K}$ and a weakening mechanism $\mathcal{W}$. $\succeq_W$ is a partial order over $\texttt{Opt}(\mathcal{K}, \mathcal{W})$.*

Proof. Straightforward from Proposition 17. □

If the user's preferences are expressed according to $\succeq_W$, then the preferred options are the least weak or, in other words, in view of Proposition 17, they are the maximal ones under set inclusion.

The third component of the framework is a mechanism for selecting the inferences to be drawn from the knowledge base. In our framework, the set of inferences is itself an option. Thus, it should be consistent. This requirement is of great importance, since it ensures that the framework delivers *safe* conclusions. Note that this inference mechanism returns an option of the language from the set of options for a given knowledge base. The set of inferences is generally computed from the preferred options. Different mechanisms can be defined for selecting the inferences to be drawn. Here is an example of such a mechanism.

Definition 24 (Universal Consequences)**.** *Let* $\langle \mathtt{Opt}(\mathcal{K}, \mathcal{W}), \succeq, \mid\!\sim \rangle$ *be a framework. A formula* $\psi \in \mathcal{L}$ *is a universal consequence of* $\mathcal{K}$ *iff* $(\forall \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}, \mathcal{W}))\psi \in \mathcal{O}$.

We can show that the set of inferences made using the universal criterion is itself an option of $\mathcal{K}$, and thus the universal criterion is a valid mechanism of inference. Moreover, it is included in every preferred option.

Proposition 18. *Let* $\langle \mathtt{Opt}(\mathcal{K}, \mathcal{W}), \succeq, \mid\!\sim \rangle$ *be a framework. The set* $\{\psi \mid \psi$ *is a universal consequence of* $\mathcal{K}\}$ *is an option in* $\mathtt{Opt}(\mathcal{L})$.

Proof. Let $C = \{\psi \mid \psi$ is a universal consequence of $\mathcal{K}\}$. As each $\mathcal{O}_i \in \mathtt{Opt}^{\succeq}(\mathcal{K}, \mathcal{W})$ is an option, $\mathcal{O}_i$ is consistent. Thus, C (which is a subset of every $\mathcal{O}_i$) is also consistent. Moreover, since $C \subseteq \mathcal{O}_i$, thus $\mathsf{CN}(C) \subseteq \mathcal{O}_i$ (by Monotonicity and Idempotence axioms), $\forall \mathcal{O}_i \in \mathtt{Opt}^{\succeq}(\mathcal{K}, \mathcal{W})$. Consequently, $\mathsf{CN}(C) \subseteq C$ (according to the above definition of universal consequences). In particular, $\mathsf{CN}(C) = C$ because of the expansion axiom. Thus, C is closed and consistent, and is therefore an option in $\mathtt{Opt}(\mathcal{L})$. □

However, the following criterion

$$\mathcal{K} \mid\!\sim \psi \text{ iff } \exists \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}, \mathcal{W}) \text{ such that } \psi \in \mathcal{O}$$

is not a valid inference mechanism since the set of consequences returned by it may be inconsistent, thus, it is not an option.

4.4 Algorithms

In this section, we present general algorithms for computing the preferred options for a given knowledge base. Throughout this section, we assume that $\mathsf{CN}(\mathcal{K})$ is finite for any knowledge base $\mathcal{K}$. The preferred options could be naively computed as follows.

```
procedure CPO-Naive($\mathcal{K}, \mathcal{W}, \succeq$)
1.  Let $X = \{\mathsf{CN}(\mathcal{K}') \mid \mathcal{K}' \in \mathcal{W}(\mathcal{K}) \wedge \mathcal{K}'$ is consistent$\}$
2.  Return any $\mathcal{O} \in X$ s.t. there is no $\mathcal{O}' \in X$ s.t. $\mathcal{O}' \succ \mathcal{O}$
```

Clearly, X is the set of options for $\mathcal{K}$. Among them, the algorithm chooses the preferred ones according to $\succeq$. Note that **CPO-Naive**, as well as the other algorithms we present in the following, relies on the CN operator, which makes the algorithm independent of the underlying logic; in order to apply the algorithm to a specific logic it suffices to provide the definition of CN for that logic. One reason for the inefficiency of **CPO-Naive** is that it makes no assumptions about the weakening mechanism and the preference relation.

The next theorem identifies the set of preferred options for a given knowledge base when $\mathcal{W}_{all}$ and $\succeq_W$ are the weakening mechanism and the preference relation, respectively.

Theorem 12. *Consider a knowledge base $\mathcal{K}$. Let $\mathcal{W}_{all}$ and $\succeq_W$ be the weakening mechanism and preference relation, respectively, that are used. Let $\Phi = \bigcup_{\psi \in \mathcal{K}} weakening(\psi)$. Then, the set of preferred options for $\mathcal{K}$ is equal to $\mathcal{PO}$ where*

$$\mathcal{PO} = \{\mathsf{CN}(\mathcal{K}') \mid \mathcal{K}' \textit{ is a maximal consistent subset of } \Phi\}$$

Proof. First, we show that any $\mathcal{O} \in \mathcal{PO}$ is a preferred option for $\mathcal{K}$. Let $\mathcal{K}'$ be a maximal consistent subset of Φ s.t. $\mathcal{O} = \mathsf{CN}(\mathcal{K}')$. It is easy to see that $\mathcal{K}'$ is in $\mathcal{W}_{all}(\mathcal{K})$. Since $\mathcal{K}'$ is consistent and $\mathcal{O} = \mathsf{CN}(\mathcal{K}')$, then $\mathcal{O}$ is an option for $\mathcal{K}$. Suppose by contradiction that $\mathcal{O}$ is not preferred, *i.e.*, there exists an option $\mathcal{O}'$ for $\mathcal{K}$ s.t. $\mathcal{O}' \succ \mathcal{O}$. Proposition 17 entails that $\mathcal{O}' \supset \mathcal{O}$. Since $\mathcal{O}'$ is an option for $\mathcal{K}$, then there exists a weakening $\mathcal{W}' \in \mathcal{W}_{all}(\mathcal{K})$ s.t. $\mathcal{O}' = \mathsf{CN}(\mathcal{W}')$. There must be a formula $\psi' \in \mathcal{W}'$ which is not in $\mathcal{O}$ (hence $\psi' \notin \mathcal{K}'$), otherwise it would be the case that $\mathcal{W}' \subseteq \mathcal{O}$ and thus $\mathsf{CN}(\mathcal{W}') \subseteq \mathcal{O}$ (from Monotonicity and Idempotence axioms), that is $\mathcal{O}' \subseteq \mathcal{O}$. Since ψ' is in a weakening of $\mathcal{K}$, then there is a (consistent) formula $\psi \in \mathcal{K}$ s.t. $\psi' \in weakening(\psi)$ and therefore $\psi' \in \Phi$. As $\mathcal{K}' \subseteq \mathcal{O} \subset \mathcal{O}'$ and $\psi' \in \mathcal{O}'$, then $\mathcal{K}' \cup \{\psi'\}$ is consistent. Since $\psi' \notin \mathcal{K}'$, $\psi' \in \Phi$, and $\mathcal{K}' \cup \{\psi'\}$ is consistent, then $\mathcal{K}'$ is not a maximal consistent subset of Φ, which is a contradiction.

We now show that every preferred option $\mathcal{O}$ for $\mathcal{K}$ is in $\mathcal{PO}$. Let $\mathcal{W}$ be a (consistent) weakening of $\mathcal{K}$ s.t. $\mathsf{CN}(\mathcal{W}) = \mathcal{O}$. It is easy to see that $\mathcal{W} \subseteq \Phi$. Then, there is a maximal consistent subset $\mathcal{K}'$ of Φ s.t. $\mathcal{W} \subseteq \mathcal{K}'$. Clearly, $\mathcal{O}' = \mathsf{CN}(\mathcal{K}')$ is in $\mathcal{PO}$, and thus, as shown above, it is a preferred option for $\mathcal{K}$. Monotonicity entails that $\mathsf{CN}(\mathcal{W}) \subseteq \mathsf{CN}(\mathcal{K}')$, that is $\mathcal{O} \subseteq \mathcal{O}'$. In particular, $\mathcal{O} = \mathcal{O}'$, otherwise Proposition 17 would entail that $\mathcal{O}$ is not preferred. □

Example 27. *Consider again the knowledge base* $\mathcal{K} = \{(a \wedge b); \neg b\}$ *of Example 20. We have that* $\Phi = \mathsf{CN}(\{a \wedge b\}) \cup \mathsf{CN}(\{\neg b\})$. *Thus, it is easy to see that a preferred option for* $\mathcal{K}$ *is* $\mathsf{CN}(\{a, \neg b\})$ *(note that* $a \in \Phi$ *since* $a \in \mathsf{CN}(\{a \wedge b\})$*).*

Clearly, we can straightforwardly derive an algorithm to compute the preferred options from the theorem above: first Φ is computed and then CN is applied to the maximal consistent subsets of Φ. Thus, such an algorithm does not need to compute all the options for a given knowledge base in order to determine the preferred ones (which is the case in the **CPO-Naive** algorithm) as every option computed by the algorithm is ensured to be preferred.

Example 28. *Consider the following inconsistent propositional Horn*[1] *knowledge base $\mathcal{K}$:*

$$\begin{aligned}
&a_1\\
&a_2 \leftarrow a_1\\
&a_3 \leftarrow a_2\\
&\vdots\\
&a_{n-1} \leftarrow a_{n-2}\\
&\neg a_1 \leftarrow a_{n-1}
\end{aligned}$$

*Suppose we want to compute one preferred option for $\mathcal{K}$ ($\mathcal{W}_{all}$ and $\succeq_W$ are the weakening mechanism and preference relation, respectively). If we use Algorithm **CPO-Naive**, then all the options for $\mathcal{K}$ w.r.t. $\mathcal{W}_{all}$ need to be computed in order to determine a preferred one. Observe that the closure of a proper subset of $\mathcal{K}$ is an option for $\mathcal{K}$, and thus the number of options is exponential. According to Theorem 12, a preferred option may be computed as $\mathsf{CN}(\mathcal{K}')$, where $\mathcal{K}'$ is a maximal consistent subset of $\bigcup_{\psi \in \mathcal{K}} weakening(\psi)$.*

Note that Theorem 12 entails that if both computing CN and consistency checking can be done in polynomial time, then one preferred option can be computed in polynomial time. For instance, this is the case for propositional Horn knowledge bases (see Section 4.5). Furthermore, observe that Theorem 12 also holds when $\supseteq$ is the preference relation simply because $\supseteq$ coincides with $\succeq_W$ (see Proposition 17).

Let us now consider the case where $\mathcal{W}_{\subseteq}$ and $\supseteq$ are the adopted weakening mechanism and preference relation, respectively.

[1] Recall that a Horn clause is a disjunction of literals containing at most one positive literal.

Theorem 13. *Consider a knowledge base $\mathcal{K}$. Let $\mathcal{W}_{\subseteq}$ and $\supseteq$ respectively be the weakening mechanism and preference relation used. Then, a knowledge base $\mathcal{O}$ is a preferred option for $\mathcal{K}$ iff $\mathcal{K}' = \mathcal{O} \cap \mathcal{K}$ is a maximal consistent subset of $\mathcal{K}$ and* $\mathsf{CN}(\mathcal{K}') = \mathcal{O}$.

Proof. ($\Leftarrow$) Clearly, $\mathcal{O}$ is an option for $\mathcal{K}$. Suppose by contradiction that $\mathcal{O}$ is not preferred, *i.e.*, there exists an option $\mathcal{O}'$ for $\mathcal{K}$ s.t. $\mathcal{O} \subset \mathcal{O}'$. Since $\mathcal{O}'$ is an option for $\mathcal{K}$, then there exists $\mathcal{W} \subseteq \mathcal{K}$ s.t. $\mathcal{O}' = \mathsf{CN}(\mathcal{W})$. There must be a formula $\psi \in \mathcal{W}$ which is not in $\mathcal{O}$ (hence $\psi \notin \mathcal{K}'$), otherwise it would be the case that $\mathcal{W} \subseteq \mathcal{O}$ and thus $\mathsf{CN}(\mathcal{W}) \subseteq \mathcal{O}$ (from Monotonicity and Idempotence axioms), that is $\mathcal{O}' \subseteq \mathcal{O}$. As $\mathcal{K}' \subseteq \mathcal{O} \subset \mathcal{O}'$ and $\psi \in \mathcal{O}'$, then $\mathcal{K}' \cup \{\psi\}$ is consistent, that is $\mathcal{K}'$ is not a maximal consistent subset of $\mathcal{K}$, which is a contradiction.

($\Rightarrow$) Suppose by contradiction that $\mathcal{O}$ is a preferred option for $\mathcal{K}$ and a case of the following occurs: (i) $\mathsf{CN}(\mathcal{K}') \neq \mathcal{O}$, (ii) $\mathcal{K}'$ is not a maximal consistent subset of $\mathcal{K}$.

(i) Since $\mathcal{K}' \subseteq \mathcal{O}$, then $\mathsf{CN}(\mathcal{K}') \subseteq \mathcal{O}$ (Monotonicity and Idempotence axioms). As $\mathsf{CN}(\mathcal{K}') \neq \mathcal{O}$, then $\mathsf{CN}(\mathcal{K}') \subset \mathcal{O}$. Since $\mathcal{O}$ is an option, then there exists $\mathcal{W} \subseteq \mathcal{K}$ s.t. $\mathcal{O} = \mathsf{CN}(\mathcal{W})$. Two cases may occur:

- $\mathcal{W} \subseteq \mathcal{K}'$. Thus, $\mathsf{CN}(\mathcal{W}) \subseteq \mathsf{CN}(\mathcal{K}')$ (Monotonicity), *i.e.*, $\mathcal{O} \subseteq \mathsf{CN}(\mathcal{K}')$, which is a contradiction.
- $\mathcal{W} \not\subseteq \mathcal{K}'$. Thus, there exists a formula ψ which is in $\mathcal{W}$ but not in $\mathcal{K}'$. Note that $\psi \in \mathcal{K}$ (as $\mathcal{W} \subseteq \mathcal{K}$) and $\psi \in \mathcal{O}$ (from the fact that $\mathcal{O} = \mathsf{CN}(\mathcal{W})$ and Expansion axiom). Since $\mathcal{K}' = \mathcal{K} \cap \mathcal{O}$, then $\psi \in \mathcal{K}'$, which is a contradiction.

(ii) Since $\mathcal{K}' \subseteq \mathcal{O}$ then $\mathcal{K}'$ is consistent and is not maximal. Thus, there exists $\mathcal{K}'' \subseteq \mathcal{K}$ which is consistent and $\mathcal{K}' \subset \mathcal{K}''$. Monotonicity implies that $\mathsf{CN}(\mathcal{K}') \subseteq \mathsf{CN}(\mathcal{K}'')$, *i.e.*, $\mathcal{O} \subseteq \mathsf{CN}(\mathcal{K}'')$ since we have proved before that $\mathcal{O} = \mathsf{CN}(\mathcal{K}')$. Let $\psi \in \mathcal{K}'' - \mathcal{K}'$. Since $\psi \in \mathcal{K}$ (as $\mathcal{K}'' \subseteq \mathcal{K}$) and $\psi \notin \mathcal{K}'$, then $\psi \notin \mathcal{O}$ (because $\mathcal{K}' = \mathcal{O} \cap \mathcal{K}$). Thus,

$\mathcal{O} \subset \mathsf{CN}(\mathcal{K}'')$. Since $\mathcal{K}''$ is consistent, then $\mathsf{CN}(\mathcal{K}'')$ is an option and $\mathcal{O}$ is not preferred, which is a contradiction.

□

The following corollary identifies the set of preferred options for a knowledge base when the weakening mechanism and the preference relation are $\mathcal{W}_{\subseteq}$ and $\supseteq$, respectively.

Corollary 3. *Consider a knowledge base $\mathcal{K}$. Let $\mathcal{W}_{\subseteq}$ and $\supseteq$ be the employed weakening mechanism and preference relation, respectively. Then, the set of preferred options for $\mathcal{K}$ is:*

$$\{\mathsf{CN}(\mathcal{K}') \mid \mathcal{K}' \mathit{\ is\ a\ maximal\ consistent\ subset\ of\ } \mathcal{K}\}$$

Proof. Straightforward from Theorem 13. □

The preceding corollary provides a way to compute the preferred options: first the maximal consistent subsets of $\mathcal{K}$ are computed, then CN is applied to them. Clearly, such an algorithm avoids the computation of every option. Note that this corollary entails that if both computing CN and consistency checking can be done in polynomial time, then one preferred option can be computed in polynomial time. Moreover, observe that both the corollary above and Theorem 13 also hold in the case where the adopted preference criterion is $\succeq_W$ because $\supseteq$ coincides with $\succeq_W$ (see Proposition 17).

We now consider the case where different assumptions on the preference relation are made. The algorithms below are independent of the weakening mechanism that we choose to use. For the sake of simplicity, we will use $\mathtt{Opt}(\mathcal{K})$ instead of $\mathtt{Opt}(\mathcal{K}, \mathcal{W})$ to denote the set of options for a knowledge base $\mathcal{K}$.

Definition 25. *A preference relation $\succeq$ is said to be* monotonic *iff for any $X, Y \subseteq \mathcal{L}$, if $X \subseteq Y$, then $Y \succeq X$. $\succeq$ is said to be* anti-monotonic *iff for any $X, Y \subseteq \mathcal{L}$, if $X \subseteq Y$, then $X \succeq Y$.*

We now define the set of minimal expansions of an option.

Definition 26. *Let $\mathcal{K}$ be a knowledge base and $\mathcal{O}$ an option for $\mathcal{K}$. We define the set of* minimal expansions *of $\mathcal{O}$ as follows:*

$$\begin{aligned} exp(\mathcal{O}) = \{\mathcal{O}' \mid \quad & \mathcal{O}' \textit{ is an option for } \mathcal{K} \wedge \\ & \mathcal{O} \subset \mathcal{O}' \wedge \\ & \textit{there does not exist an option } \mathcal{O}'' \textit{ for } \mathcal{K} \textit{ s.t. } \mathcal{O} \subset \mathcal{O}'' \subset \mathcal{O}'\} \end{aligned}$$

Given a set S of options, we define $exp(S) = \bigcup_{\mathcal{O} \in S} exp(\mathcal{O})$.

Clearly, the way $exp(\mathcal{O})$ is computed depends on the adopted weakening mechanism. In the following algorithm, the preference relation $\succeq$ is assumed to be anti-monotonic.

```
procedure CPO-Anti($\mathcal{K}, \succeq$)
1.  $S_0 = \{\mathcal{O} \mid \mathcal{O}$ is a minimal (under $\subseteq$) option for $\mathcal{K}\}$
2.  Construct a maximal sequence $S_1, \ldots, S_n$ s.t. $S_i \neq \emptyset$ where
    $S_i = \{\mathcal{O} \mid \mathcal{O} \in exp(S_{i-1}) \wedge \nexists \mathcal{O}' \in S_0(\mathcal{O}' \subset \mathcal{O} \wedge \mathcal{O} \not\succeq \mathcal{O}')\}, 1 \leq i \leq n$
3.  $S = \bigcup_{i=0}^{n} S_i$
4.  Return the $\succeq$-preferred options in $S$
```

Clearly, the algorithm always terminates, since each option in S_i is a proper superset of some option in S_{i-1} and the size of an option for $\mathcal{K}$ is bounded. The algorithm exploits the anti-monotonicity of $\succeq$ to reduce the set of options from which the preferred ones are determined. First, the algorithm computes the minimal options for $\mathcal{K}$. Then, the algorithm computes bigger and bigger options and the anti-monotonicity of $\succeq$ is used to discard those options that are not preferred for sure: when S_i is computed, we consider every minimal expansion $\mathcal{O}$ of some option in S_{i-1}; if $\mathcal{O}$ is a proper superset of an option $\mathcal{O}' \in S_0$ and $\mathcal{O} \not\succeq \mathcal{O}'$, then $\mathcal{O}$ can be discarded since $\mathcal{O}' \succeq \mathcal{O}$ by the anti-monotonicity of $\succeq$ and therefore $\mathcal{O}' \succ \mathcal{O}$ (note that any option that is a superset of $\mathcal{O}$ will be discarded as well).

Observe that in the worst case the algorithm has to compute every option for $\mathcal{K}$ (*e.g.*, when $\mathcal{O}_1 \succeq \mathcal{O}_2$ for any $\mathcal{O}_1, \mathcal{O}_2 \in \texttt{Opt}(\mathcal{K})$ as in this case every option is preferred).

Example 29. *Consider the following knowledge base $\mathcal{K}$ containing check-in times for the employees in a company for a certain day.*

ψ_1 $\quad checkedIn\_Mark\_9AM$

ψ_2 $\quad checkedIn\_Claude\_8AM$

ψ_3 $\quad checkedIn\_Mark\_10AM$

ψ_4 $\quad \neg(checkedIn\_Mark\_9AM \wedge checkedIn\_Mark\_10AM)$

Formula ψ_1 and ψ_2 state that employee Mark *and* Claude *checked in for work at 9 AM and 8 AM, respectively. However, formula ψ_3 records that employee* Mark *checked in for work at 10 AM that day. Furthermore, as it is not possible for a person to check in for work at different times on the same day, we also have formula ψ_4, which is the instantiation of that constraint for employee* Mark.

*Assume that each formula ψ_i has an associated non-negative weight $w_i \in [0, 1]$ corresponding to the likelihood of the formula being wrong, and suppose those weights are $w_1 = 0.2, w_2 = 0, w_3 = 0.1,$ and $w_4 = 0$. Suppose that the weight of an option $\mathcal{O}$ is $w(\mathcal{O}) = \sum_{\psi_i \in \mathcal{K} \cap \mathcal{O}} w_i$. Let $\mathcal{W}_{\subseteq}$ be the weakening mechanism used, and consider the preference relation defined as follows: $\mathcal{O}_i \succeq \mathcal{O}_j$ iff $w(\mathcal{O}_i) \leq w(\mathcal{O}_j)$. Clearly, the preference relation is anti-monotonic. Algorithm **CPO-Anti** first computes $S_0 = \{\mathcal{O}_0 =$ $\mathsf{CN}(\emptyset)\}$. It then looks for the minimal expansions of $\mathcal{O}_0$ which are preferable to $\mathcal{O}_0$. In* *this case, we have $\mathcal{O}_1 = \mathsf{CN}(\{\psi_2\})$ and $\mathcal{O}_2 = \mathsf{CN}(\{\psi_4\})$; hence, $S_1 = \{\mathcal{O}_1, \mathcal{O}_2\}$. Note that neither $\mathsf{CN}(\{\psi_1\})$ nor $\mathsf{CN}(\{\psi_3\})$ is preferable to $\mathcal{O}_0$ and thus they can be discarded because $\mathcal{O}_0$ turns out to be strictly preferable to them. The algorithm then looks for the minimal expansions of some option in S_1 which are preferable to $\mathcal{O}_0$; the only one is $\mathcal{O}_3 = \mathsf{CN}(\{\psi_2, \psi_4\})$, so $S_3 = \{\mathcal{O}_3\}$. It is easy to see that S_4 is empty and thus the algorithm returns the preferred options from those in $S_0 \cup S_1 \cup S_2 \cup S_3$, which are $\mathcal{O}_0$, $\mathcal{O}_1$, $\mathcal{O}_2$, and $\mathcal{O}_3$. Note that the algorithm avoided the computation of every option for $\mathcal{K}$.*

We now show the correctness of the algorithm.

Theorem 14. *Let $\mathcal{K}$ be a knowledge base and $\succeq$ an anti-monotonic preference relation. Then,*

- *(Soundness) If **CPO-Anti**$(\mathcal{K}, \succeq)$ returns $\mathcal{O}$, then $\mathcal{O}$ is a preferred option for $\mathcal{K}$.*
- *(Completeness) For any preferred option $\mathcal{O}$ for $\mathcal{K}$, $\mathcal{O}$ is returned by **CPO-Anti**$(\mathcal{K}, \succeq)$.*

Proof. Let S be the set of options for $\mathcal{K}$ computed by the algorithm. First of all, we show that for any option $\mathcal{O}' \in \mathtt{Opt}(\mathcal{K}) - S$ there exists an option $\mathcal{O}'' \in S$ s.t. $\mathcal{O}'' \succ \mathcal{O}'$. Suppose by contradiction that there is an option $\mathcal{O}' \in \mathtt{Opt}(\mathcal{K}) - S$ s.t. there does not exist an option $\mathcal{O}'' \in S$ s.t. $\mathcal{O}'' \succ \mathcal{O}'$. Since $\mathcal{O}' \notin S_0$, then $\mathcal{O}'$ is not a minimal option for $\mathcal{K}$. Hence, there exist an option $\mathcal{O}_0 \in S_0$ and $n \geq 0$ options $\mathcal{O}_1, \ldots, \mathcal{O}_n$ s.t. $\mathcal{O}_0 \subset \mathcal{O}_1 \subset \cdots \subset \mathcal{O}_n \subset \mathcal{O}_{n+1} = \mathcal{O}'$ and $\mathcal{O}_i \in exp(\mathcal{O}_{i-1})$ for $1 \leq i \leq n+1$. Since $\nexists \mathcal{O}'' \in S_0$ s.t. $\mathcal{O}'' \succ \mathcal{O}'$, then $\nexists \mathcal{O}'' \in S_0$ s.t. $\mathcal{O}'' \succ \mathcal{O}_i$ for $0 \leq i \leq n$, otherwise $\mathcal{O}'' \succ \mathcal{O}_i$ and $\mathcal{O}_i \succeq \mathcal{O}'$ (by anti-monotonicity of $\succeq$) would imply $\mathcal{O}'' \succ \mathcal{O}'$, which is a contradiction. It can be easily verified, by induction on i, that $\mathcal{O}_i \in S_i$ for $0 \leq i \leq n+1$, and then $\mathcal{O}' \in S_{n+1}$, which is a contradiction.

(Soundness). Clearly, $\mathcal{O}$ is an option for $\mathcal{K}$. Suppose by contradiction that $\mathcal{O}$ is not preferred, *i.e.*, there exists an option $\mathcal{O}'$ for $\mathcal{K}$ s.t. $\mathcal{O}' \succ \mathcal{O}$. Clearly, $\mathcal{O}' \in \mathtt{Opt}(\mathcal{K}) - S$, otherwise it would be the case that $\mathcal{O}' \in S$ and then $\mathcal{O}$ is not returned by the algorithm (see step 4). We have proved above that there exists $\mathcal{O}'' \in S$ s.t. $\mathcal{O}'' \succ \mathcal{O}'$. Since $\mathcal{O}'' \succ \mathcal{O}'$ and $\mathcal{O}' \succ \mathcal{O}$, then $\mathcal{O}'' \succ \mathcal{O}$ (by the transitivity of $\succ$), which is a contradiction (as $\mathcal{O}, \mathcal{O}'' \in S$ and $\mathcal{O}$ is a $\succeq$-preferred option in S).

(Completeness). Suppose by contradiction that $\mathcal{O}$ is not returned by the algorithm. Clearly, this means that $\mathcal{O} \in \mathtt{Opt}(\mathcal{K}) - S$. We have proved above that this implies that there exists an option $\mathcal{O}'' \in S$ s.t. $\mathcal{O}'' \succ \mathcal{O}$, which is a contradiction. □

Observe that when the adopted weakening mechanism is either $\mathcal{W}_{\subseteq}$ or $\mathcal{W}_{all}$, the first step becomes $S_0 = \{\mathsf{CN}(\emptyset)\}$, whereas the second step can be specialized as follows: $S_i = \{\mathcal{O} \mid \mathcal{O} \in exp(S_{i-1}) \ \wedge \ \mathcal{O} \succeq \mathsf{CN}(\emptyset)\}$.

We now consider the case where $\succeq$ is assumed to be monotonic.

Definition 27. *Let $\mathcal{K}$ be a knowledge base and $\mathcal{O}$ an option for $\mathcal{K}$. We define the set of* minimal contractions *of $\mathcal{O}$ as follows:*

$$\begin{aligned} contr(\mathcal{O}) = \{\mathcal{O}' \mid \quad & \mathcal{O}' \textit{ is an option for } \mathcal{K} \wedge \\ & \mathcal{O}' \subset \mathcal{O} \wedge \\ & \textit{there does not exist an option } \mathcal{O}'' \textit{ for } \mathcal{K} \textit{ s.t. } \mathcal{O}' \subset \mathcal{O}'' \subset \mathcal{O}\}. \end{aligned}$$

Given a set S of options, we define $contr(S) = \bigcup_{\mathcal{O} \in S} contr(\mathcal{O})$.

Observe that how to compute $contr(\mathcal{O})$ depends on the considered weakening mechanism. In the following algorithm the preference relation $\succeq$ is assumed to be monotonic.

```
procedure CPO-Monotonic($\mathcal{K}, \succeq$)
1. $S_0 = \{\mathcal{O} \mid \mathcal{O}$ is a maximal (under $\subseteq$) option for $\mathcal{K}\}$;
2. Construct a maximal sequence $S_1, \dots, S_n$ s.t. $S_i \neq \emptyset$ where
   $S_i = \{\mathcal{O} \mid \mathcal{O} \in \mathit{contr}(S_{i-1}) \wedge \nexists \mathcal{O}' \in S_0(\mathcal{O} \subset \mathcal{O}' \wedge \mathcal{O} \not\succeq \mathcal{O}')\}, 1 \leq i \leq n$
3. $S = \bigcup_{i=0}^{n} S_i$
4. Return the $\succeq$-preferred options in $S$.
```

Clearly, the algorithm always terminates, since each option in S_i is a proper subset of some option in S_{i-1}. The algorithm exploits the monotonicity of $\succeq$ to reduce the set of options from which the preferred ones are determined. The algorithm first computes the maximal (under $\subseteq$) options for $\mathcal{K}$. It then computes smaller and smaller options and the monotonicity of $\succeq$ is used to discard those options that are not preferred for sure: when S_i is computed, we consider every minimal contraction $\mathcal{O}$ of some option in S_{i-1}; if $\mathcal{O}$ is a proper subset of an option $\mathcal{O}' \in S_0$ and $\mathcal{O} \not\succeq \mathcal{O}'$, then $\mathcal{O}$ can be discarded since $\mathcal{O}' \succeq \mathcal{O}$ by the monotonicity of $\succeq$ and therefore $\mathcal{O}' \succ \mathcal{O}$. Note that any option that is a subset of $\mathcal{O}$ will be discarded as well.

Observe that in the worst case the algorithm has to compute every option for $\mathcal{K}$ (*e.g.*, when $\mathcal{O}_1 \succeq \mathcal{O}_2$ for any $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K})$ as in this case every option is preferred).

It is worth noting that when the adopted weakening mechanism is $\mathcal{W}_{all}$, the first step of the algorithm can be implemented by applying Theorem 12 since it identifies the options which are maximal under set inclusion (recall that $\succeq_W$ coincides with $\supseteq$, see Proposition 17). Likewise, when the weakening mechanism is $\mathcal{W}_{\subseteq}$, the first step of the algorithm can be accomplished by applying Corollary 3.

Example 30. *Consider again the knowledge base of Example 29. Suppose now that each formula ψ_i has associated a non-negative weight $w_i \in [0, 1]$ corresponding to the* *reliability of the formula, and let those weights be $w_1 = 0.1, w_2 = 1, w_3 = 0.2$, and $w_4 = 1$. Once again, the weight of an option $\mathcal{O}$ is $w(\mathcal{O}) = \sum_{\psi_i \in \mathcal{K} \cap \mathcal{O}} w_i$. Let $\mathcal{W}_{\subseteq}$ be the weakening mechanism, and consider the preference relation defined as follows: $\mathcal{O}_i \succeq \mathcal{O}_j$ iff $w(\mathcal{O}_i) \geq w(\mathcal{O}_j)$. Clearly, the preference relation is monotonic. Algorithm **CPO-Monotonic** first computes the maximal options,* i.e., $S_0 = \{\mathcal{O}_1 = \mathsf{CN}(\{\psi_2, \psi_3, \psi_4\}),$ $\mathcal{O}_2 = \mathsf{CN}(\{\psi_1, \psi_2, \psi_4\}), \mathcal{O}_4 = \mathsf{CN}(\{\psi_1, \psi_2, \psi_3\})\}$. *After that, the algorithm looks for a minimal contraction $\mathcal{O}$ of some option in S_0 s.t. there is no superset $\mathcal{O}' \in S_0$ of $\mathcal{O}$ s.t. $\mathcal{O} \not\succeq \mathcal{O}'$. It is easy to see that in this case there is no option that satisfies this property,* i.e., $S_1 = \emptyset$. *Thus, the algorithm returns the preferred options in S_0, namely $\mathcal{O}_1$. Note that the algorithm avoided the computation of every option for $\mathcal{K}$.*

We now show the correctness of the algorithm.

Theorem 15. *Let $\mathcal{K}$ be a knowledge base and $\succeq$ a monotonic preference relation. Then,*

- *(Soundness) If **CPO-Monotonic**$(\mathcal{K}, \succeq)$ returns $\mathcal{O}$, then $\mathcal{O}$ is a preferred option for $\mathcal{K}$.*
- *(Completeness) For any preferred option $\mathcal{O}$ for $\mathcal{K}$, $\mathcal{O}$ is returned by **CPO-Monotonic**$(\mathcal{K}, \succeq)$.*

Proof. Let S be the set of options for $\mathcal{K}$ computed by the algorithm. First of all, we show that for any option $\mathcal{O}' \in \mathtt{Opt}(\mathcal{K}) - S$, there exists an option $\mathcal{O}'' \in S$ s.t. $\mathcal{O}'' \succ \mathcal{O}'$. Suppose by contradiction that there is an option $\mathcal{O}' \in \mathtt{Opt}(\mathcal{K}) - S$ s.t. there does not exist an option $\mathcal{O}'' \in S$ s.t. $\mathcal{O}'' \succ \mathcal{O}'$. Since $\mathcal{O}' \notin S_0$, then $\mathcal{O}'$ is not a maximal option for $\mathcal{K}$. Hence, there exist an option $\mathcal{O}_0 \in S_0$ and $n \geq 0$ options $\mathcal{O}_1, \ldots, \mathcal{O}_n$ s.t. $\mathcal{O}_0 \supset \mathcal{O}_1 \supset \cdots \supset \mathcal{O}_n \supset \mathcal{O}_{n+1} = \mathcal{O}'$ and $\mathcal{O}_i \in \mathit{contr}(\mathcal{O}_{i-1})$ for $1 \leq i \leq n+1$. Since $\nexists \mathcal{O}'' \in S_0$ s.t. $\mathcal{O}'' \succ \mathcal{O}'$, then $\nexists \mathcal{O}'' \in S_0$ s.t. $\mathcal{O}'' \succ \mathcal{O}_i$ for $0 \leq i \leq n$, otherwise $\mathcal{O}'' \succ \mathcal{O}_i$ and $\mathcal{O}_i \succeq \mathcal{O}'$ (by monotonicity of $\succeq$) would imply $\mathcal{O}'' \succ \mathcal{O}'$, which is a contradiction.

It can be easily verified, by induction on i, that $\mathcal{O}_i \in S_i$ for $0 \leq i \leq n+1$, and then $\mathcal{O}' \in S_{n+1}$, which is a contradiction.

The soundness and completeness of the algorithm can be shown in the same way as in the proof of Theorem 14. □

4.5 Handling Inconsistency in Monotonic Logics

In this section, we consider several monotonic logics and show how our framework is well-suited to handle inconsistency in these logics. It is particularly important to note that reasoning about inconsistency in many of these logics has not been studied before. As a consequence, our general framework for reasoning about inconsistency is not only new, it also yields new algorithms and new results for such reasoning in existing logics. We also study the complexity of the universal inference problem for many of these logics.

4.5.1 Propositional Horn-clause Logic

Let us consider knowledge bases consisting of propositional Horn clauses. Recall that a Horn Clause is an expression of the form $L_1 \vee \cdots \vee L_n$ where each L_i is a propositional literal such that *at most* one L_i is positive.[2] We will assume that the consequences of a knowledge base are those determined by the application of modus ponens.

Proposition 19. *Consider a propositional Horn knowledge base $\mathcal{K}$. Let $\mathcal{W}_{\subseteq}$ and $\supseteq$ respectively be the weakening mechanism and preference relation that are used. A preferred option for $\mathcal{K}$ can be computed in polynomial time.*

Proof. Corollary 3 entails that a preferred option can be computed by finding a maximal consistent subset $\mathcal{K}'$ of $\mathcal{K}$ and then computing $\mathsf{CN}(\mathcal{K}')$. Since both checking consistency

[2] Note that a definite clause is a Horn clause where exactly one L_i is positive. It is well known that any set of definite clauses is always consistent.

and computing consequences can be accomplished in polynomial time [Pap94], the overall computation is in polynomial time. □

Nevertheless, the number of preferred options may be exponential, as shown in the following example.

Example 31. *Consider the propositional Horn knowledge base*

$$\mathcal{K} = \{a_1, \neg a_1, \dots, a_n, \neg a_n\}$$

containing $2n$ formulas, where the a_i's are propositional variables. It is easy to see that the set of preferred options for $\mathcal{K}$ is

$$\{\mathsf{CN}(\{l_1, \dots, l_n\}) \mid l_i \in \{a_i, \neg a_i\} \textit{ for } i = 1..n\}$$

whose cardinality is 2^n ($\mathcal{W}_{\subseteq}$ and $\supseteq$ are, respectively, the weakening mechanism and preference relation used).

The following theorem addresses the complexity of computing universal consequences of propositional Horn knowledge bases.

Proposition 20. *Let $\mathcal{K}$ and ψ be a propositional Horn knowledge base and clause, respectively. Let $\mathcal{W}_{\subseteq}$ and $\supseteq$ respectively be a weakening mechanism and preference relation. The problem of deciding whether ψ is a universal consequence of $\mathcal{K}$ is coNP-complete.*

Proof. It follows from Corollary 3 and the result in [CLS94] stating that the problem of deciding whether a propositional Horn formula is a consequence of every maximal consistent subset of a Horn knowledge base is coNP-complete. □

Note that when the weakening mechanism and the preference relation are $\mathcal{W}_{all}$ and $\succeq_W$, respectively, both the set of options and preferred options do not differ from those obtained when $\mathcal{W}_{\subseteq}$ and $\supseteq$ are considered. In fact, since $weakening(\psi) = \{\psi\}$ for any propositional Horn formula ψ, then $\mathcal{W}_{\subseteq}$ and $\mathcal{W}_{all}$ are the same. Proposition 17 states that $\supseteq$ and $\succeq_W$ coincide. Thus, the previous results are trivially extended to the case where $\mathcal{W}_{all}$ and $\succeq_W$ are considered.

Corollary 4. *Consider a propositional Horn knowledge base $\mathcal{K}$. Let $\mathcal{W}_{all}$ and $\succeq_W$ respectively be the weakening mechanism and preference relation that are used. A preferred option for $\mathcal{K}$ can be computed in polynomial time.*

Proof. Follows immediately from Proposition 19. □

Corollary 5. *Let $\mathcal{K}$ and ψ be a propositional Horn knowledge base and clause, respectively. Let $\mathcal{W}_{all}$ and $\succeq_W$ respectively be the weakening mechanism and preference relation that are used. The problem of deciding whether ψ is a universal consequence of $\mathcal{K}$ is coNP-complete.*

Proof. Follows immediately from Proposition 20. □

4.5.2 Propositional Probabilistic Logic

In this section, we consider the probabilistic logic of [Nil86a] extended to probability intervals, *i.e.*, formulas are of the form $\phi : [\ell, u]$, where ϕ is a classical propositional formula and $[\ell, u]$ is a subset of the real unit interval.

The existence of a set of propositional symbols is assumed. A *world* is any set of propositional symbols; we use W to denote the set of all possible worlds. A *probabilistic interpretation* I is a probability distribution over worlds, *i.e.*, it is a function $I : W \rightarrow [0, 1]$ such that $\sum_{w \in W} I(w) = 1$. Then, I *satisfies* a formula $\phi : [\ell, u]$ iff $\ell \leq \sum_{w \in W, w \models \phi} I(w) \leq u$. Consistency and entailment are defined in the standard way.

Example 32. *Consider a network of sensors collecting information about people's positions. Suppose the following knowledge base $\mathcal{K}$ is obtained by merging information collected by different sensors.*

$$\begin{array}{ll} \psi_1 & loc\_John\_X : [0.6, 0.7] \\ \psi_2 & loc\_John\_X \vee loc\_John\_Y : [0.3, 0.5] \end{array}$$

The first formula in $\mathcal{K}$ says that $John$'s position is X with a probability between 0.6 and 0.7. The second formula states that $John$ is located either in position X or in position Y with a probability between 0.3 and 0.5. The knowledge base above is inconsistent: since every world in which the first formula is true satisfies the second formula as well, the probability of the latter has to be greater than or equal to the probability of the former.

As already mentioned before, a reasonable weakening mechanism for probabilistic knowledge bases consists of making probability intervals wider.

Definition 28. *For any probabilistic knowledge base $\mathcal{K} = \{\phi_1 : [\ell_1, u_1], \ldots, \phi_n : [\ell_n, u_n]\}$, the weakening mechanism $\mathcal{W}_P$ is defined as follows: $\mathcal{W}_P(\mathcal{K}) = \{\{\phi_1 : [\ell'_1, u'_1], \ldots, \phi_n : [\ell'_n, u'_n]\} \mid [\ell_i, u_i] \subseteq [\ell'_i, u'_i], 1 \leq i \leq n\}$.*

Example 33. *Consider again the probabilistic knowledge base $\mathcal{K}$ of Example 32. The weakenings of $\mathcal{K}$ determined by $\mathcal{W}_P$ are of the form:*

$$\begin{array}{ll} \psi'_1 & loc\_John\_X : [\ell_1, u_1] \\ \psi'_2 & loc\_John\_X \vee loc\_John\_Y : [\ell_2, u_2] \end{array}$$

where $[0.6, 0.7] \subseteq [\ell_1, u_1]$ and $[0.3, 0.5] \subseteq [\ell_2, u_2]$. The options for $\mathcal{K}$ (w.r.t. $\mathcal{W}_P$) are the closure of those weakenings s.t. $[\ell_1, u_1] \cap [\ell_2, u_2] \neq \emptyset$ (this condition ensures consistency).

Suppose that the preferred options are those that modify the probability intervals as little as possible: $\mathcal{O}_i \succeq_P \mathcal{O}_j$ *iff* $sc(\mathcal{O}_i) \leq sc(\mathcal{O}_j)$ *for any options* $\mathcal{O}_i, \mathcal{O}_j$ *for* $\mathcal{K}$*, where* $sc(\mathsf{CN}(\{\psi_1', \psi_2'\})) =$ *diff*$(\psi_1, \psi_1') +$ *diff*(ψ_2, ψ_2') *and diff*$(\phi : [\ell_1, u_1], \phi : [\ell_2, u_2]) = \ell_1 - \ell_2 + u_2 - u_1$*. The preferred options are the closure of:*

$$loc\_John\_X : [\ell, 0.7]$$

$$loc\_John\_X \vee loc\_John\_Y : [0.3, \ell]$$

where $0.5 \leq \ell \leq 0.6$*.*

We now define the preference relation introduced in the example above.

Definition 29. *Let* $\mathcal{K} = \{\phi_1 : [\ell_1, u_1], \ldots, \phi_n : [\ell_n, u_n]\}$ *be a probabilistic knowledge base. We say that the score of an option* $\mathcal{O} = \mathsf{CN}(\{\phi_1 : [\ell_1', u_1'], \ldots, \phi_n : [\ell_n', u_n']\})$ *in* $\mathtt{Opt}(\mathcal{K}, \mathcal{W}_P)$ *is* $sc(\mathcal{O}) = \sum_{i=1}^{n}(\ell_i - \ell_i') + (u_i' - u_i)$*. We define the preference relation* $\succeq_P$ *as follows: for any* $\mathcal{O}, \mathcal{O}' \in \mathtt{Opt}(\mathcal{K}, \mathcal{W}_P)$*,* $\mathcal{O} \succeq_P \mathcal{O}'$ *iff* $sc(\mathcal{O}) \leq sc(\mathcal{O}')$*.*

The weakenings (under $\mathcal{W}_P$) whose closure yields the preferred options (w.r.t. $\succeq_P$) can be found by solving a linear program derived from the original knowledge base. We now show how to derive such a linear program.

In the following definition we use W to denote the set of possible worlds for a knowledge base $\mathcal{K}$, that is, $W = 2^{\Sigma}$, Σ being the set of propositional symbols appearing in $\mathcal{K}$.

Definition 30. *Let* $\mathcal{K} = \{\phi_1 : [\ell_1, u_1], \ldots, \phi_n : [\ell_n, u_n]\}$ *be a probabilistic knowledge base. Then,* $\mathsf{LP}(\mathcal{K})$ *is the following linear program:*

$$
\begin{aligned}
&\textbf{\textit{minimize}} \sum_{i=1}^{n} (\ell_i - \ell'_i) + (u'_i - u_i) \\
&\textbf{\textit{subject to}} \\
&\qquad \ell'_i \leq \textstyle\sum_{w \in W, w \models \phi_i} p_w \leq u'_i, \quad 1 \leq i \leq n \\
&\qquad \textstyle\sum_{w \in W} p_w = 1 \\
&\qquad 0 \leq \ell'_i \leq \ell_i, \quad 1 \leq i \leq n \\
&\qquad u_i \leq u'_i \leq 1, \quad 1 \leq i \leq n
\end{aligned}
$$

Clearly, in the definition above, the ℓ'_i's, u_i's and p_w's are variables (p_w denotes the probability of world w). We denote by $Sol(\mathsf{LP}(\mathcal{K}))$ the set of solutions of $\mathsf{LP}(\mathcal{K})$. We also associate a knowledge base $\mathcal{K}_{\mathcal{S}}$ to every solution $\mathcal{S}$ as follows: $\mathcal{K}_{\mathcal{S}} = \{\phi_i : [\mathcal{S}(\ell'_i), \mathcal{S}(u'_i)] \mid 1 \leq i \leq n\}$, where $\mathcal{S}(x)$ is the value assigned to variable x by solution $\mathcal{S}$. Intuitively, the knowledge base $\mathcal{K}_{\mathcal{S}}$ is the knowledge base obtained by setting the bounds of each formula in $\mathcal{K}$ to the values assigned by solution $\mathcal{S}$.

The following theorem states that the solutions of the linear program $\mathsf{LP}(\mathcal{K})$ derived from a knowledge base $\mathcal{K}$ "correspond to" the preferred options of $\mathcal{K}$ when the weakening mechanism is $\mathcal{W}_P$ and the preference relation is $\succeq_P$.

Theorem 16. *Given a probabilistic knowledge base* $\mathcal{K}$*,*

1. *if* $\mathcal{S} \in Sol(\mathsf{LP}(\mathcal{K}))$*, then* $\exists \mathcal{O} \in \mathtt{Opt}^{\succeq_P}(\mathcal{K}, \mathcal{W}_P)$ *s.t.* $\mathcal{O} = \mathsf{CN}(\mathcal{K}_{\mathcal{S}})$*,*

2. *if* $\mathcal{O} \in \mathtt{Opt}^{\succeq_P}(\mathcal{K}, \mathcal{W}_P)$*, then* $\exists \mathcal{S} \in Sol(\mathsf{LP}(\mathcal{K}))$ *s.t.* $\mathcal{O} = \mathsf{CN}(\mathcal{K}_{\mathcal{S}})$*.*

Proof. Let LP' be the linear program obtained from $\mathsf{LP}(\mathcal{K})$ by discarding the objective function.

(a) We first show that if $\mathcal{S} \in Sol(\mathsf{LP}')$, then $\exists \mathcal{O} \in \mathtt{Opt}(\mathcal{K}, \mathcal{W}_P)$ s.t. $\mathcal{O} = \mathsf{CN}(\mathcal{K}_\mathcal{S})$. Clearly, $\mathcal{K}_\mathcal{S} \in \mathcal{W}_P(\mathcal{K})$ as the third and fourth sets of constraints in LP' ensure that $[\ell_i, u_i] \subseteq [\ell'_i, u'_i]$ for any $\phi_i : [\ell_i, u_i] \in \mathcal{K}$. The first and second sets of constraints in LP' ensure that $\mathcal{K}_\mathcal{S}$ is consistent – a model for $\mathcal{K}_\mathcal{S}$ is simply given by the p_w's. Thus, $\mathsf{CN}(\mathcal{K}_\mathcal{S})$ is an option for $\mathcal{K}$.

(b) We now show that if $\mathcal{O} \in \mathtt{Opt}(\mathcal{K}, \mathcal{W}_P)$, then $\exists \mathcal{S} \in Sol(\mathsf{LP}')$ s.t. $\mathcal{O} = \mathsf{CN}(\mathcal{K}_\mathcal{S})$. Since $\mathcal{O}$ is an option, then there exists $\mathcal{K}' \in \mathcal{W}_P(\mathcal{K})$ s.t. $\mathcal{O} = \mathsf{CN}(\mathcal{K}')$. Clearly, $\mathcal{K}'$ is consistent. Let I be a model of $\mathcal{K}'$. It is easy to see that if we assign p_w to $I(w)$ for every world w and the ℓ'_i's and u'_i's are assigned to the bounds of ϕ_i in $\mathcal{K}'$, then such an assignment satisfies every constraint of LP'.

It is easy to see that given a solution $\mathcal{S}$ of LP', the value of the objective function of $\mathsf{LP}(\mathcal{K})$ for $\mathcal{S}$ is exactly the score sc assigned to the option $\mathsf{CN}(\mathcal{K}_\mathcal{S})$ by $\succeq_P$ (see Definition 29).

1. Suppose that $\mathcal{S} \in Sol(\mathsf{LP}(\mathcal{K}))$. As shown above, since $\mathcal{S}$ satisfies the constraints of $\mathsf{LP}(\mathcal{K})$, then there exists an option $\mathcal{O}$ s.t. $\mathcal{O} = \mathsf{CN}(\mathcal{K}_\mathcal{S})$. Suppose by contradiction that $\mathcal{O}$ is not preferred, that is, there is another option $\mathcal{O}'$ s.t. $sc(\mathcal{O}') < sc(\mathcal{O})$. Then, there is a solution $\mathcal{S}'$ of LP' s.t. $\mathcal{O}' = \mathsf{CN}(\mathcal{K}_{\mathcal{S}'})$. Since the objective function of $\mathsf{LP}(\mathcal{K})$ corresponds to sc, then $\mathcal{S}$ does not minimize the objective function, which is a contradiction.

2. Suppose that $\mathcal{O} \in \mathtt{Opt}^{\succeq_P}(\mathcal{K}, \mathcal{W}_P)$. As shown above, since $\mathcal{O}$ is an option, then there exists a solution $\mathcal{S}$ of LP' s.t. $\mathcal{O} = \mathsf{CN}(\mathcal{K}_\mathcal{S})$. Suppose by contradiction that $\mathcal{S}$ is not a solution of $\mathsf{LP}(\mathcal{K})$. This means that it does not minimize the objective function. Then, there is a solution $\mathcal{S}'$ of $\mathsf{LP}(\mathcal{K})$ which has a lower value of the objective function. As shown before, $\mathcal{O}' = \mathsf{CN}(\mathcal{K}_{\mathcal{S}'})$ is an option and has a score lower than $\mathcal{O}$, which is a contradiction.

□

We refer to probabilistic knowledge bases whose formulas are built from propositional Horn formulas as *Horn probabilistic knowledge bases*. The following theorem states that already for this restricted subset of probabilistic logic, the problem of deciding whether a formula is a universal consequence of a knowledge base is coNP-hard.

Theorem 17. *Let $\mathcal{K}$ and ψ be a Horn probabilistic knowledge base and formula, respectively. Suppose that the weakening mechanism returns subsets of the given knowledge base and the preference relation is $\supseteq$. The problem of deciding whether ψ is a universal consequence of $\mathcal{K}$ is coNP-hard.*

Proof. We reduce 3-DNF VALIDITY to our problem. Let $\phi = C_1 \vee \cdots \vee C_n$ be an instance of 3-DNF VALIDITY, where the C_i's are conjunctions containing exactly three literals, and X the set of propositional variables appearing in ϕ. We derive from ϕ a Horn probabilistic knowledge base $\mathcal{K}^*$ as follows. Given a literal ℓ of the form x (resp. $\neg x$), with $x \in X$, we denote with $p(\ell)$ the propositional variable x^T (resp. x^F). Let

$$\mathcal{K}_1 = \{u \leftarrow p(\ell_1) \wedge p(\ell_2) \wedge p(\ell_3) : [1,1] \mid \ell_1 \wedge \ell_2 \wedge \ell_3 \; is\ a\ conjunction\ of\ \phi\}$$

and

$$\mathcal{K}_2 = \{u \leftarrow x^T \wedge x^F : [1,1] \mid x \in X\}$$

Given a variable $x \in X$, let

$$\begin{aligned} \mathcal{K}_x = \{ \quad & x^T : [1,1], \\ & x^F : [1,1], \\ & \leftarrow x^T \wedge x^F : [1,1]\} \end{aligned}$$

Finally,

$$\mathcal{K}^* = \mathcal{K}_1 \cup \mathcal{K}_2 \cup \bigcup_{x \in X} \mathcal{K}_x$$

The derived instance of our problem is $(\mathcal{K}^*, u : [1, 1])$. First of all, note that $\mathcal{K}^*$ is inconsistent since $\mathcal{K}_x$ is inconsistent for any $x \in X$. The set of maximal consistent subsets of $\mathcal{K}^*$ is:

$$\mathcal{M} = \left\{ \mathcal{K}_1 \cup \mathcal{K}_2 \cup \bigcup_{x \in X} \mathcal{K}'_x \mid \mathcal{K}'_x \; is \; a \; maximal \; consistent \; subset \; of \; \mathcal{K}_x \right\}$$

Note that a maximal consistent subset of $\mathcal{K}_x$ is obtained from $\mathcal{K}_x$ by discarding exactly one formula. Corollary 3 entails that the set of preferred options for $\mathcal{K}^*$ is $\mathtt{Opt}^{\succeq}(\mathcal{K}^*) = \{\mathsf{CN}(S) \mid S \in \mathcal{M}\}$. We partition $\mathtt{Opt}^{\succeq}(\mathcal{K}^*)$ into two sets: $\mathcal{O}_1 = \{\mathcal{O} \mid \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}^*) \wedge \exists x \in X \; s.t. \; x^T : [1, 1], x^F : [1, 1] \in \mathcal{O}\}$ and $\mathcal{O}_2 = \mathtt{Opt}^{\succeq}(\mathcal{K}^*) - \mathcal{O}_1$. We now show that ϕ is valid iff $u : [1, 1]$ is a universal consequence of $\mathcal{K}^*$.

($\Rightarrow$) It is easy to see that every preferred option $\mathcal{O}$ in $\mathcal{O}_1$ contains $u : [1, 1]$, since there exists $x \in X$ s.t. $x^T : [1, 1], x^F : [1, 1] \in \mathcal{O}$ and $u \leftarrow x^T \wedge x^F : [1, 1] \in \mathcal{O}$. Consider now a preferred option $\mathcal{O} \in \mathcal{O}_2$. For any $x \in X$ either $x^T : [1, 1]$ or $x^F : [1, 1]$ belongs to $\mathcal{O}$. Let us consider the truth assignment I derived from $\mathcal{O}$ as follows: for any $x \in X$, $I(x)$ is true iff $x^T : [1, 1] \in \mathcal{O}$ and $I(x)$ is false iff $x^F : [1, 1] \in \mathcal{O}$. Since ϕ is valid, then I satisfies ϕ, *i.e.*, there is a conjunction $\ell_1 \wedge \ell_2 \wedge \ell_3$ of ϕ which is satisfied by I. It is easy to see that $u : [1, 1]$ can be derived from the rule $u \leftarrow p(\ell_1) \wedge p(\ell_2) \wedge p(\ell_3) : [1, 1]$ in $\mathcal{K}_1$. Hence, $u : [1, 1]$ is a universal consequence of $\mathcal{K}^*$.

($\Leftarrow$) We show that if ϕ is not valid then there exists a preferred option $\mathcal{O}$ for $\mathcal{K}^*$ s.t. $u : [1, 1] \notin \mathcal{O}$. Consider a truth assignment for ϕ which does not satisfy ϕ and let $True$ and $False$ be the set of variables of ϕ made true and false, respectively, by such an assignment. Consider the following set

$$\begin{aligned} S = \ & \mathcal{K}_1 \ \cup \ \mathcal{K}_2 \\ & \cup \textstyle\bigcup_{x \in True} \{x^T : [1,1], \leftarrow x^T \wedge x^F : [1,1]\} \\ & \cup \textstyle\bigcup_{x \in False} \{x^F : [1,1], \leftarrow x^T \wedge x^F : [1,1]\} \end{aligned}$$

It is easy to see that S is a maximal consistent subset of $\mathcal{K}^*$, and thus $\mathcal{O} = \mathsf{CN}(S)$ is a preferred option for $\mathcal{K}^*$. It can be easily verified that $u : [1,1] \notin \mathcal{O}$. □

4.5.3 Propositional Linear Temporal Logic

Temporal logic has been extensively used for reasoning about programs and their executions. It has achieved a significant role in the formal specification and verification of concurrent and distributed systems [Pnu77]. In particular, a number of useful concepts such as safety, liveness and fairness can be formally and concisely specified using temporal logics [MP92, Eme90].

In this section, we consider *Propositional Linear Temporal Logic* (PLTL) [GPSS80] - a logic used in verification of systems and reactive systems. Basically, this logic extends classical propositional logic with a set of temporal connectives. The particular variety of temporal logic we consider is based on a linear, discrete model of time isomorphic to the natural numbers. Thus, the temporal connectives operate over a sequence of distinct "moments" in time. The connectives that we consider are $\Diamond$ (*sometime in the future*), $\Box$ (*always in the future*) and $\bigcirc$ (*at the next point in time*).

Assuming a countable set Σ of propositional symbols, every $p \in \Sigma$ is a PLTL formula. If ϕ and ψ are PLTL formulas, then the following are PLTL formulas as well: $\phi \vee \psi,\ \phi \wedge \psi,\ \neg\phi,\ \phi \leftarrow \psi,\ \Box\phi,\ \bigcirc\phi,\ \Diamond\phi$

The notion of a timeline can be formalized with a function $I : \mathbb{N} \rightarrow 2^{\Sigma}$ that maps each natural number (representing a moment in time) to a set of propositional symbols (intuitively, this is the set of propositional symbols which are true at that moment). We say that

- $(I, i) \models p$ iff $p \in I(i)$, where $p \in \Sigma$;
- $(I, i) \models \bigcirc\phi$ iff $(I, i+1) \models \phi$;
- $(I, i) \models \Diamond\phi$ iff $\exists j.\, j \geq i \wedge (I, j) \models \phi$;
- $(I, i) \models \Box\phi$ iff $\forall j.\, j \geq i \; implies \; (I, j) \models \phi$.

The semantics for the standard connectives is as expected. I is a model of a PLTL formula ϕ iff $(I, 0) \models \phi$. Consistency and entailment are defined in the standard way.

Example 34. *Consider the PLTL knowledge base reported below [Art08] which specifies the behavior of a computational system.*

$$\begin{array}{ll} \psi_1 & \Box(\Diamond \mathit{received} \leftarrow \mathit{requested}) \\ \psi_2 & \Box(\bigcirc \mathit{processed} \leftarrow \mathit{received}) \end{array}$$

The first statement says that it is always the case that if a request is issued, then it will be received at some future time point. The second statement says that it is always the case that if a request is received, then it will be processed at the next time point. The statements above correspond to the definition of the system, i.e., *how the system is supposed to behave. Suppose now that there is a monitoring system which reports data regarding the system's behavior and, for instance, the following formula is added to the knowledge base:*

$$\psi_3 \qquad \mathit{received} \wedge \bigcirc\neg \mathit{processed} \wedge \bigcirc \bigcirc \mathit{processed}$$

The inclusion of ψ_3 makes the knowledge base inconsistent, since the monitoring system is reporting that a request was received and was not processed at the next moment in time, but two moments afterwards.

Consider a weakening function that replaces the $\bigcirc$ operator with the $\Diamond$ operator in a formula ψ, provided that the formula thus obtained is a consequence of ψ. Suppose that preferred options are those that keep as many monitoring system formulas unchanged (i.e. unweakened) as possible. In this case, the only preferred option is $\mathsf{CN}(\{\psi_1, \Box(\Diamond processed \leftarrow received), \psi_3\})$*, where formula ψ_2, stating that if a request is received then it will be processed at the next moment in time, has been weakened into a new one stating that the request will be processed at a future point in time.*

Theorem 18. *Let $\mathcal{K}$ and ψ be a temporal knowledge base and formula, respectively. Suppose the weakening mechanism returns subsets of the given knowledge base and the preference relation is $\supseteq$. The problem of deciding whether ψ is a universal consequence of $\mathcal{K}$ is coNP-hard.*

Proof. A reduction from 3-DNF VALIDITY to our problem can be carried out in a similar way to the the proof of Theorem 17. Let $\phi = C_1 \vee \cdots \vee C_n$ be an instance of 3-DNF VALIDITY, where the C_i's are conjunctions containing exactly three literals, and X the set of propositional variables appearing in ϕ. We derive from ϕ a temporal knowledge base $\mathcal{K}^*$ as follows. Given a literal ℓ of the form x (resp. $\neg x$), with $x \in X$, we denote with $p(\ell)$ the propositional variable x^T (resp. x^F). Let

$$\mathcal{K}_1 = \{\Box\,(u \leftarrow p(\ell_1) \wedge p(\ell_2) \wedge p(\ell_3)) \mid \ell_1 \wedge \ell_2 \wedge \ell_3 \; is\ a\ conjunction\ of\ \phi\}$$

and

$$\mathcal{K}_2 = \{\Box\,(u \leftarrow x^T \wedge x^F) \mid x \in X\}$$

Given a variable $x \in X$, let

$$\begin{aligned} \mathcal{K}_x = \{ \quad & \Box\, x^T, \\ & \Box\, x^F, \\ & \Box\, (\leftarrow x^T \wedge x^F)\} \end{aligned}$$

Finally,

$$\mathcal{K}^* = \mathcal{K}_1 \cup \mathcal{K}_2 \cup \bigcup_{x \in X} \mathcal{K}_x$$

The derived instance of our problem is $(\mathcal{K}^*, \Box\, u)$. The claim can be proved in a similar way to the proof of Theorem 17. □

4.5.4 Fuzzy Logic

In this section we consider fuzzy logic. Formulas are of the form $\phi : v$, where ϕ is a propositional formula built from a set Σ of propositional symbols and the logical connectives $\neg, \wedge, \vee$, and $v \in [0, 1]$ (we call v the *degree of truth*). An interpretation I assigns a value in $[0, 1]$ to each propositional symbol in Σ. Moreover, given two propositional formulas ϕ_1 and ϕ_2, we have

- $I(\neg\phi_1) = 1 - I(\phi_1)$
- $I(\phi_1 \wedge \phi_2) = min\{I(\phi_1), I(\phi_2)\}$
- $I(\phi_1 \vee \phi_2) = max\{I(\phi_1), I(\phi_2)\}$

We say that I satisfies a formula $\phi : v$ iff $I(\phi) \geq v$. Consistency and entailment are defined in the standard way.

Example 35. *Consider the following inconsistent knowledge base:*

$$\begin{aligned} \psi_1 : &\quad a : 0.7 \\ \psi_2 : &\quad b : 0.6 \\ \psi_3 : &\quad \neg(a \wedge b) : 0.5 \end{aligned}$$

Suppose that the weakening mechanism is defined as follows: for any formula $\phi : v$, we define $\mathcal{W}(\phi : v) = \{\phi : v' | \; v' \in [0,1] \wedge v' \leq v\}$; then, $\mathcal{W}(\mathcal{K}) = \{\{\psi_1', \psi_2', \psi_3'\} \mid \psi_i' \in \mathcal{W}(\psi_i) \; 1 \leq i \leq 3\}$. Thus, options are the closure of

$$\begin{aligned} \psi_1 : &\quad a : v_1 \\ \psi_2 : &\quad b : v_2 \\ \psi_3 : &\quad \neg(a \wedge b) : v_3 \end{aligned}$$

where $v_1 \leq 0.7$, $v_2 \leq 0.6$, $v_3 \leq 0.5$, and $1 - min\{v_1, v_2\} \geq v_3$. Suppose that the preferred options are those that modify a minimum number of formulas. Thus, the preferred options are of the form

$$\begin{aligned} &\mathsf{CN}(\{a : v_1, \psi_2, \psi_3\}) \textit{ where } v_1 \leq 0.5, \textit{or} \\ &\mathsf{CN}(\{b : v_2, \psi_1, \psi_3\}) \textit{ where } v_2 \leq 0.5, \textit{or} \\ &\mathsf{CN}(\{\neg(a \wedge b) : v_3, \psi_1, \psi_2\}) \textit{ where } v_3 \leq 0.4 \end{aligned}$$

Finally, suppose that the preference relation is expressed as before but, in addition, we would like to change the degree of truth as little as possible. In this case, the preferred options are:

$$\begin{aligned} &\mathsf{CN}(\{b : 0.5, \psi_1, \psi_3\}) \\ &\mathsf{CN}(\{\neg(a \wedge b) : 0.4, \psi_1, \psi_2\}) \end{aligned}$$

We refer to fuzzy knowledge bases whose formulas are built from propositional Horn formulas as *Horn fuzzy knowledge bases*. The following theorem states that even for this restricted subset of fuzzy logic, the problem of deciding whether a formula is a universal consequence of a knowledge base is coNP-hard.

Theorem 19. *Let $\mathcal{K}$ and ψ be a Horn fuzzy knowledge base and formula, respectively. Let $\mathcal{W}_{\subseteq}$ and $\supseteq$ be the adopted weakening mechanism and preference relation, respectively. The problem of deciding whether ψ is a universal consequence of $\mathcal{K}$ is coNP-hard.*

Proof. A reduction from 3-DNF VALIDITY to our problem can be carried out in a way similar to the the proof of Theorem 17. Let $\phi = C_1 \vee \cdots \vee C_n$ be an instance of 3-DNF VALIDITY, where the C_i's are conjunctions containing exactly three literals, and X is the set of propositional variables appearing in ϕ. We derive from ϕ a Horn temporal knowledge base $\mathcal{K}^*$ as follows. Given a literal ℓ of the form x (resp. $\neg x$), with $x \in X$, we denote with $p(\ell)$ the propositional variable x^T (resp. x^F). Let

$$\mathcal{K}_1 = \{u \vee \neg p(\ell_1) \vee \neg p(\ell_2) \vee \neg p(\ell_3) : 1 \mid \ell_1 \wedge \ell_2 \wedge \ell_3 \; is \; a \; conjunction \; of \; \phi\}$$

and

$$\mathcal{K}_2 = \{u \vee \neg x^T \vee \neg x^F : 1 | \, x \in X\}$$

Given a variable $x \in X$, let

$$\begin{aligned} \mathcal{K}_x = \{ \quad & x^T : 1, \\ & x^F : 1, \\ & \neg x^T \vee \neg x^F : 1\} \end{aligned}$$

Finally,

$$\mathcal{K}^* = \mathcal{K}_1 \cup \mathcal{K}_2 \cup \bigcup_{x \in X} \mathcal{K}_x$$

The derived instance of our problem is $(\mathcal{K}^*, u : 1)$. The claim can be proved in a way similar to the proof of Theorem 17. □

4.5.5 Belief Logic

In this section we focus on the belief logic presented in [Lev84]. Formulas are formed from a set Σ of primitive propositions, the standard connectives $\vee$, $\wedge$, and $\neg$, and two unary connectives B and L. Neither B nor L appear within the scope of the other. Connective B is used to express what is *explicitly* believed by an agent (a sentence that is actively held to be true by the agent), whereas L is used to express what is *implicitly* believed by the agent (*i.e.*, the consequences of his explicit beliefs).

Semantics of sentences is given in terms of a model structure $\langle S, \mathcal{B}, T, F\rangle$, where *S* is a set of *situations*, $\mathcal{B}$ is a subset of *S* (the situations that could be the actual ones according to what is believed), and *T* and *F* are functions from Σ to subsets of *S*. Intuitively, $T(p)$ are the situations that support the truth of p and $F(p)$ are the situations that support the falsity of p. A primitive proposition may be true, false, both, or neither in a situation. A *complete* situation (or *possible world*) is one that supports either the truth or the falsity (not both) of every primitive proposition. A complete situation s is *compatible* with a situation s' if s and s' agree whenever s' is defined, *i.e.*, if $s' \in T(p)$ then $s \in T(p)$, and if $s' \in F(p)$ then $s \in F(p)$, for each primitive proposition p. Let $\mathcal{W}(\mathcal{B})$ consist of all complete situations in S compatible with some situation in $\mathcal{B}$.

Two *support relations* $\models_T$ and $\models_F$ between situations and formulas are defined in the following way:

- $s \models_T p$ iff $s \in T(p)$, where p is a primitive proposition ,

- $s \models_F p$ iff $s \in F(p)$, where p is a primitive proposition ;

- $s \models_T (\alpha \vee \beta)$ iff $s \models_T \alpha$ or $s \models_T \beta$,
- $s \models_F (\alpha \vee \beta)$ iff $s \models_F \alpha$ and $s \models_F \beta$;
- $s \models_T (\alpha \wedge \beta)$ iff $s \models_T \alpha$ and $s \models_T \beta$,
- $s \models_F (\alpha \wedge \beta)$ iff $s \models_F \alpha$ or $s \models_F \beta$;
- $s \models_T \neg\alpha$ iff $s \models_F \alpha$,
- $s \models_F \neg\alpha$ iff $s \models_T \alpha$;
- $s \models_T B\alpha$ iff for every $s' \in \mathcal{B}, s' \models_T \alpha$,
- $s \models_F B\alpha$ iff $s \not\models_T B\alpha$;
- $s \models_T L\alpha$ iff for every $s' \in \mathcal{W}(\mathcal{B}),\ s' \models_T \alpha$,
- $s \models_F L\alpha$ iff $s \not\models_T L\alpha$.

Given a complete situation s in S, then if $s \models_T \alpha$, then α is true at s, otherwise α is said to be false at s. Finally, α is said to be valid ($\models \alpha$) iff for any model structure $\langle \mathit{S}, \mathit{B}, \mathit{T}, \mathit{F} \rangle$ and any complete situation s in S, α is true at s. The satisfiability of a sentence is defined analogously; entailment is defined in the expected way.

Note that belief logic allows an agent to believe contradictory sentences, *e.g.*, $\{Bp, B\neg p\}$ is a consistent knowledge base. However, $\{Bp, \neg Bp\}$ is inconsistent.

Example 36. *Consider the following inconsistent knowledge base $\mathcal{K}$ that represents the knowledge of an agent regarding a city's subway system:*

$$\begin{aligned}
\psi_1 &: \ \textit{goingNorthTrain1} \\
\psi_2 &: \ B\ \textit{goingNorthTrain1} \\
\psi_3 &: \ \textit{goingNorthTrain1} \rightarrow \textit{canGetUpTownFromStationA} \\
\psi_4 &: \ B(\textit{goingNorthTrain1} \rightarrow \textit{canGetUpTownFromStationA}) \\
\psi_5 &: \ \neg(\textit{canGetUpTownFromStationA})
\end{aligned}$$

Using a train schedule associated with train station A, we might be able to express formulas ψ_1 and ψ_3. ψ_1 states that Train 1 *goes north, whereas ψ_3 states that if* Train 1 *goes north, then the agent can get uptown from station A. Formulas ψ_2 and ψ_4 state that the agent explicitly believes in the information that he got from the schedule. However, this knowledge base is inconsistent because of the presence of formula ψ_5, which states that it is not possible to get uptown from station A, for instance, because that route is closed for repairs.*

Suppose that each formula ψ_i is associated with a time stamp $t(\psi_i)$ that represents the moment in time in which the agent acquired that piece of information. In this case, we consider the subsets of $\mathcal{K}$ as its weakenings, and the preference relation is defined in such a way that maximal (under $\subseteq$) options are preferable to the others, and among these we say that $\mathcal{O}_i \succeq \mathcal{O}_j$ iff $sc(\mathcal{O}_i) \geq sc(\mathcal{O}_j)$ where $sc(\mathcal{O}) = \sum_{\psi \in \mathcal{O} \cap \mathcal{K}} t(\psi)$, i.e.*, we would like to preserve as many formulas as possible and more up to date information. If in our example we have $t(\psi_1) = t(\psi_2) = 1$, $t(\psi_3) = t(\psi_4) = 3$, and $t(\psi_5) = 5$, then the only preferred option is $\mathsf{CN}(\{\psi_2, \psi_3, \psi_4, \psi_5\})$.*

Theorem 20. *Let $\mathcal{K}$ and ψ be a belief knowledge base and formula, respectively. Suppose that the weakening mechanism returns subsets of the given knowledge base and the* *preference relation is $\supseteq$. The problem of deciding whether ψ is a universal consequence of $\mathcal{K}$ is coNP-hard.*

Proof. A reduction from 3-DNF VALIDITY to our problem can be carried out in a similar way to the the proof of Theorem 17 by using only propositional formulas. Let $\phi = C_1 \vee \cdots \vee C_n$ be an instance of 3-DNF VALIDITY, where the C_i's are conjunctions containing exactly three literals, and X is the set of propositional variables appearing in ϕ. We derive from ϕ a belief knowledge base $\mathcal{K}^*$ as follows. Given a literal ℓ of the form x (resp. $\neg x$), with $x \in X$, we denote by $p(\ell)$ the propositional variable x^T (resp. x^F). Let

$$\mathcal{K}_1 = \{u \vee \neg p(\ell_1) \vee \neg p(\ell_2) \vee \neg p(\ell_3) \mid \ell_1 \wedge \ell_2 \wedge \ell_3 \; is \; a \; conjunction \; of \; \phi\}$$

and

$$\mathcal{K}_2 = \{u \vee \neg x^T \vee \neg x^F \mid x \in X\}$$

Given a variable $x \in X$, let

$$\begin{aligned} \mathcal{K}_x = \{ \quad & x^T, \\ & x^F, \\ & \neg x^T \vee \neg x^F\} \end{aligned}$$

Finally,

$$\mathcal{K}^* = \mathcal{K}_1 \cup \mathcal{K}_2 \cup \bigcup_{x \in X} \mathcal{K}_x$$

The derived instance of our problem is $(\mathcal{K}^*, u)$. The claim can be proved in a similar way to the proof of Theorem 17. □

4.5.6 Spatial Logic

In this section we consider the Region Connection Calculus (RCC) proposed in [RCC92]. This logic is a topological approach to qualitative spatial representation and reasoning (see [CR08]) where *spatial regions* are subsets of a topological space. Relationships between spatial regions are defined in first order logic in terms of a primitive binary relation $C(x, y)$, which means "x connects with y"; this relation is reflexive and symmetric and holds when regions x and y share a common point. **RCC-8** considers the following eight *base relations*: DC (*disconnected*), EC (*externally connected*), PO (*partial overlap*), EQ (*equal*), TPP (*tangential proper part*), $NTPP$ (*non-tangential proper part*), TPP^{-1} (*the inverse of TPP*), $NTPP^{-1}$ (*the inverse of NTPP*). These relations are jointly exhaustive and pairwise disjoint, *i.e.*, exactly one of them holds between any two spatial regions. Figure 4.1 shows two-dimensional examples of the 8 basic relations. If the exact relation is not known, we can also use the union of different base relations, *e.g.*, $a\,\{NTPP, NTPP^{-1}\}\,b$ means that either a is a non-tangential proper part of b or vice versa.

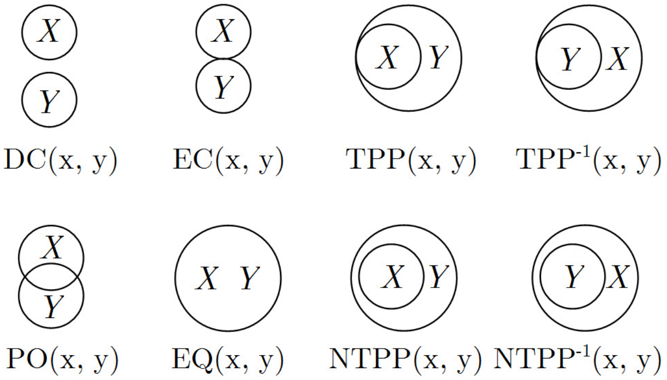


Figure 4.1: Two-dimensional examples for **RCC-8** base relations.

Example 37. *Consider the following knowledge base $\mathcal{K}$:*

$$\begin{array}{ll} \psi_1 & a\,\{NTPP\}\,b \\ \psi_2 & b\,\{EC\}\,c \\ \psi_3 & a\,\{EC\}\,c \end{array}$$

The knowledge base is inconsistent since the first two formulas imply that a and c are disconnected, whereas the last one states that they are externally connected.

Suppose the weakening mechanism used is $\mathcal{W}_{all}$. In this case, the knowledge base is weakened by making its formulas more undefined. For instance, some options for $\mathcal{K}$ are

$$\begin{aligned} \mathcal{O}_1 &= \mathsf{CN}(\{a\,\{NTPP, TPP\}\,b, \psi_2, \psi_3\}) \\ \mathcal{O}_2 &= \mathsf{CN}(\{b\,\{EC, PO, NTPP^{-1}\}\,c, \psi_1, \psi_3\}) \\ \mathcal{O}_3 &= \mathsf{CN}(\{b\,\{EC, NTPP^{-1}\}\,c, \psi_1, \psi_3\}) \\ \mathcal{O}_4 &= \mathsf{CN}(\{a\,\{NTPP, TPP\}\,b, b\,\{EC, DC\}\,c, \psi_3\}) \end{aligned}$$

Suppose the preference relation chooses those options that weaken a minimum number of formulas as the preferred options. In this case, $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{O}_3$ are preferred options, whereas $\mathcal{O}_4$ is not.

Suppose the preference relation is $\succeq_W$. Then $\mathcal{O}_1$ and $\mathcal{O}_3$ are preferred options, whereas $\mathcal{O}_2$ and $\mathcal{O}_4$ are not. In fact, it is easy to see that $\mathcal{O}_3 \succeq_W \mathcal{O}_2$ but not vice versa, and $\mathcal{O}_1 \succeq_W \mathcal{O}_4$ but not vice versa.

*Finally, suppose that options that only weaken formulas of the form $x\,\{NTPP\}\,y$ or $x\,\{TPP\}\,y$ into $x\,\{NTPP, TPP\}\,y$ are preferable to the others (*e.g.*, because we are not sure if a region is a tangential or non-tangential proper part of another, but the information about the other topological relations is reliable and we would prefer not to change it). In this case, $\mathcal{O}_1$ is a preferred option whereas the others are not.*

4.6 Link with Existing Approaches

As discussed in Section 2.1, two kinds of approaches have been proposed in the literature for solving the problem of inconsistency. The first type focuses on revising the knowledge base and restoring its consistency. The second approach accepts inconsistency and copes with it, prohibiting the logic from deriving trivial inferences. We have described some of the works in the literature for both types of approaches in Section 2.1. In this section we will analyze some of the works in the literature for the first approach and establish the relationship with the general framework for inconsistency management proposed in this chapter.

We will start analyzing the proposal of [RM70]; this work considers propositional knowledge bases. Preferences among the maximal consistent subsets of the original knowledge may be expressed, so that *preferred* maximal consistent subsets are determined. A formula is a *P-consequence* of a possibly inconsistent knowledge base $\mathcal{K}$, where *P* is the preference criterion, if it is a consequence of every preferred (according to *P*) maximal consistent subset of $\mathcal{K}$. The paper discusses various preference criteria. Given a knowledge base $\mathcal{K}$ and a preference criterion *P*, we use $MCS(\mathcal{K}, P)$ to denote the set of preferred maximal consistent subsets of $\mathcal{K}$.

Definition 31. *Consider a knowledge base $\mathcal{K}$ and a preference criterion P on the maximal consistent subsets of $\mathcal{K}$. Suppose $\mathcal{W}_{\subseteq}$ is the weakening mechanism used. For any $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K})$, we say that $\mathcal{O}_1 \succeq \mathcal{O}_2$ iff there exists $\mathcal{K}_1 \in MCS(\mathcal{K}, P)$ s.t. $\mathcal{O}_1 = \mathsf{CN}(\mathcal{K}_1)$.*

As stated in the following proposition, the preferred maximal consistent subsets *correspond to* the preferred options when the weakening mechanism is $\mathcal{W}_{\subseteq}$ and the preference relation is the one of the definition above.

Proposition 21. *Let $\mathcal{K}$ be a knowledge base and P a preference criterion on the maximal consistent subsets of $\mathcal{K}$. Let $\mathcal{W}_{\subseteq}$ be the adopted weakening mechanism and $\succeq$ the preference relation of Definition 31. Then:*

- $\forall S \in MCS(\mathcal{K}, P),\ \exists \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K})$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.
- $\forall \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}),\ \exists S \in MCS(\mathcal{K}, P)$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.

Proof. Straightforward. □

Clearly, P-consequences correspond to our notion of universal consequence (Definition 24). Note that our framework is not restricted to propositional knowledge bases only and gives the flexibility to choose a weakening mechanism different from $\mathcal{W}_{\subseteq}$.

We focus now on prioritized knowledge bases, such as those in [Bre89]. We will analyze two generalizations of Poole's approach [Poo88]. In the following, a knowledge base is a set of classical first order formulas. In the first generalization, a knowledge base $\mathcal{K}$ is supposed to be stratified into $\mathcal{K}_1, \ldots, \mathcal{K}_n$ ($\mathcal{K} = \mathcal{K}_1 \cup \ldots \cup \mathcal{K}_n$) such that the formulas in the same stratum are equally preferred, whereas formulas in a stratum $\mathcal{K}_i$ are preferred to formulas in $\mathcal{K}_j$ with $i < j$.

Definition 32. *([Bre89]) Let $\mathcal{K} = \mathcal{K}_1 \cup \ldots \cup \mathcal{K}_n$ be a knowledge base. $S = S_1 \cup \ldots \cup S_n$ is a* preferred subbase *of $\mathcal{K}$ if and only if $\forall j, 1 \leq j \leq n$, $S_1 \cup \ldots \cup S_j$ is a maximal (under set-inclusion) consistent subset of $\mathcal{K}_1 \cup \ldots \cup \mathcal{K}_j$.*
$P_1(\mathcal{K})$ denotes the set of preferred subbases of $\mathcal{K}$.

We show that the approach above can be captured by our framework by defining the appropriate weakening mechanism and preference relation.

Definition 33. *Consider a knowledge base $\mathcal{K}$ and let $\mathcal{W}_{\subseteq}$ be the adopted weakening mechanism. For any $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K})$, we say that $\mathcal{O}_1 \succeq \mathcal{O}_2$ iff there exists $\mathcal{K}_1 \in P_1(\mathcal{K})$ s.t. $\mathcal{O}_1 = \mathsf{CN}(\mathcal{K}_1)$.*

Proposition 22. *Let $\mathcal{K}$ be a knowledge base, $\mathcal{W}_{\subseteq}$ the weakening mechanism and $\succeq$ the preference relation of Definition 33. Then,*

- $\forall S \in P_1(\mathcal{K}),\ \exists \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K})$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.
- $\forall \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}),\ \exists S \in P_1(\mathcal{K})$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.

Proof. Straightforward. □

The second generalization is based on a partial order on the formulas of a knowledge base.

Definition 34. *Let $<$ be a strict partial order on a knowledge base $\mathcal{K}$. S is a* preferred subbase *of $\mathcal{K}$ if and only if there exists a strict total order $\psi_1, \ldots, \psi_n$ of $\mathcal{K}$ respecting $<$ such that $S = S_n$ with*

$$S_0 = \emptyset$$

$$S_i = \begin{cases} S_{i-1} \cup \{\psi_i\} & if\ S_{i-1} \cup \{\psi_i\}\ is\ consistent \\ S_{i-1} & otherwise \end{cases} \qquad 1 \leq i \leq n$$

$P_2(\mathcal{K})$ denotes the set of preferred subbases of $\mathcal{K}$.

In addition, the second generalization can be easily expressed in our framework.

Definition 35. *Consider a knowledge base $\mathcal{K}$ and let $\mathcal{W}_{\subseteq}$ be the adopted weakening mechanism. For any $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K})$, we say that $\mathcal{O}_1 \succeq \mathcal{O}_2$ iff there exists $\mathcal{K}_1 \in P_2(\mathcal{K})$ s.t. $\mathcal{O}_1 = \mathsf{CN}(\mathcal{K}_1)$.*

Proposition 23. *Let $\mathcal{K}$ be a knowledge base, $\mathcal{W}_{\subseteq}$ the weakening mechanism used, and $\succeq$ the preference relation of Definition 35. Then,*

- $\forall S \in P_2(\mathcal{K}),\ \exists \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K})$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.
- $\forall \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}),\ \exists S \in P_2(\mathcal{K})$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.

Proof. Straightforward. □

Brewka [Bre89] provides a *weak* and *strong* notion of provability for both the generalizations described above. A formula ψ is *weakly provable* from a knowledge base $\mathcal{K}$ iff there is a preferred subbase S of $\mathcal{K}$ s.t. $\psi \in \mathsf{CN}(S)$; ψ is *strongly provable* from $\mathcal{K}$ iff for every preferred subbase S of $\mathcal{K}$ we have $\psi \in \mathsf{CN}(S)$. Clearly, the latter notion of provability corresponds to our notion of universal consequence (Definition 24), whereas the former is not a valid inference mechanism, since the set of weakly provable formulas might be inconsistent. Observe that Brewka's approach is committed to a specific logic, weakening mechanism and preference criterion, whereas our framework is applicable to different logics and gives the flexibility to choose the weakening mechanism and the preference relation that the end-user believes more suitable for his purposes.

Looking at inconsistency management approaches based on a partial order on the formulas of a knowledge, the work of [Roo92] proposes the concept of a *reliability theory*, based on a partial *reliability relation* among the formulas in a first order logic knowledge base $\mathcal{K}$. Clearly, this approach can be expressed in our framework in a manner analogous to Definition 35 for Brewka's approach. The author defines a special purpose logic based on first order calculus, and a deduction process to obtain the set of premises that can be *believed* to be true. The deduction process is based on the computation of justifications (premisses used in the derivation of contradictions) for believing or removing formulas, and it iteratively constructs and refines these justifications. At each step, the set of formulas that can be believed from a set of justifications can be computed in time $O(n * m)$ where n is the number of justifications used in that step and m is the number of formulas in the theory.

Finally, we focus on priority-based management of inconsistent knowledge bases, as in [BCD$^+$93, CLS95]. Propositional knowledge bases are considered and a knowledge base $\mathcal{K}$ is supposed to be stratified into strata $\mathcal{K}_1, \ldots, \mathcal{K}_n$, where $\mathcal{K}_1$ consists of the formulas of highest priority whereas $\mathcal{K}_n$ contains the formulas of lowest priority. Priorities in $\mathcal{K}$ are used to select preferred consistent subbases. Inferences are made from the preferred subbases of $\mathcal{K}$, that is $\mathcal{K}$ entails a formula ψ iff ψ can be classically inferred from every preferred subbase of $\mathcal{K}$. The work in [BCD$^+$93] presents different meaning of "preferred", which are reported in the following definition.

Definition 36. *([BCD$^+$93]) Let $\mathcal{K} = (\mathcal{K}_1 \cup \cdots \cup \mathcal{K}_n)$ be a propositional knowledge base, $X = (X_1 \cup \cdots \cup X_n)$ and $Y = (Y_1 \cup \cdots \cup Y_n)$ two consistent subbases of $\mathcal{K}$, where $X_i = X \cap \mathcal{K}_i$ and $Y_i = Y \cap \mathcal{K}_i$. We define:*

- best-out preference*: let $a(Z) = min\{i \mid \exists\psi \in \mathcal{K}_i - Z\}$ for a consistent subbase Z of $\mathcal{K}$, with the convention $min\,\emptyset = n + 1$. The best-out preference is defined by $X \ll^{bo} Y$ iff $a(X) \leq a(Y)$;*

- inclusion-based preference *is defined by $X \ll^{incl} Y$ iff $\exists i\ s.t.\ X_i \subset Y_i$ and $\forall j\ s.t.\ 1 \leq j < i, X_j = Y_j$;*

- lexicographic preference *is defined by $X \ll^{lex} Y$ iff $\exists i\ s.t.\ |X_i| < |Y_i|$ and $\forall j\ s.t.\ 1 \leq j < i, |X_j| = |Y_j|$.*

Let us consider the best-out preference and let $amax(\mathcal{K}) = max\{i \mid \mathcal{K}_1 \cup \cdots \cup \mathcal{K}_i\ is\ consistent\}$. If $amax(\mathcal{K}) = k$, then the *best-out preferred* consistent subbases of

$\mathcal{K}$ are exactly the consistent subbase of $\mathcal{K}$ which contain $(\mathcal{K}_1 \cup \cdots \cup \mathcal{K}_k)$; we denote them by $P_{bo}(\mathcal{K})$. This approach can be easily captured by our framework by adopting $\mathcal{W}_{\subseteq}$ as weakening mechanism and defining the preference relation as follows.

Definition 37. *Consider a knowledge base $\mathcal{K}$ and let $\mathcal{W}_{\subseteq}$ be the adopted weakening mechanism. For any $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K})$, we say that $\mathcal{O}_1 \succeq \mathcal{O}_2$ iff there exists $\mathcal{K}_1 \in P_{bo}(\mathcal{K})$ s.t. $\mathcal{O}_1 = \mathsf{CN}(\mathcal{K}_1)$.*

Proposition 24. *Let $\mathcal{K}$ be a knowledge base, $\mathcal{W}_{\subseteq}$ the weakening mechanism and $\succeq$ the preference relation of Definition 37. Then,*

- $\forall S \in P_{bo}(\mathcal{K}),\ \exists \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K})$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.
- $\forall \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}),\ \exists S \in P_{bo}(\mathcal{K})$ *such that* $\mathcal{O}$ = $\mathsf{CN}(S)$.

Proof. Straightforward. □

The *inclusion-based preferred* subbases are of the form $(X_1 \cup \cdots \cup X_n)$ s.t. $(X_1 \cup \cdots \cup X_i)$ is a maximal (under set inclusion) consistent subbase of $(\mathcal{K}_1 \cup \cdots \cup \mathcal{K}_i)$, for $i = 1..n$. Note these preferred subbases coincide with Brewka's preferred subbases of Definition 32 above, which can be expressed in our framework.

Finally, the lexicographic preferred subbases are of the form $(X_1 \cup \cdots \cup X_n)$ s.t. $(X_1 \cup \cdots \cup X_i)$ is a cardinality-maximal consistent subbase of $(\mathcal{K}_1 \cup \cdots \cup \mathcal{K}_i)$, for $i = 1..n$; we denote them by $P_{lex}(\mathcal{K})$.

Definition 38. *Consider a knowledge base $\mathcal{K}$ and let $\mathcal{W}_{\subseteq}$ be the adopted weakening mechanism. For any $\mathcal{O}_1, \mathcal{O}_2 \in \mathtt{Opt}(\mathcal{K})$, we say that $\mathcal{O}_1 \succeq \mathcal{O}_2$ iff there exists $\mathcal{K}_1 \in P_{lex}(\mathcal{K})$ s.t. $\mathcal{O}_1 = \mathsf{CN}(\mathcal{K}_1)$.*

Proposition 25. *Let $\mathcal{K}$ be a knowledge base, $\mathcal{W}_{\subseteq}$ the weakening mechanism and $\succeq$ the preference relation of Definition 38. Then,*

- $\forall S \in P_{lex}(\mathcal{K}),\ \exists \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K})$ *such that* $\mathcal{O} = \mathsf{CN}(S)$.

- $\forall \mathcal{O} \in \mathtt{Opt}^{\succeq}(\mathcal{K}),\ \exists S \in P_{lex}(\mathcal{K})$ *such that* $\mathcal{O}$ = $\mathsf{CN}(S)$.

Proof. Straightforward. □

As already said before, once a criterion for determining preferred subbase has been fixed, a formula is a consequence of $\mathcal{K}$ if can be classically inferred from every preferred subbase, which corresponds to our universal inference mechanism (Definition 24).

In [CLS95], the same criteria for selecting preferred consistent subbases are considered, and three entailment principles are presented. The *UNI principle* is the same as in [BCD^{+}93], i.e. it corresponds to our universal inference mechanism. According to the *EXI principle*, a formula ψ is inferred from a knowledge base $\mathcal{K}$ if ψ is classically inferred from at least one preferred subbase of $\mathcal{K}$. According to the *ARG principle*, a formula ψ is inferred from a knowledge base $\mathcal{K}$ if ψ is classically inferred from at least one preferred subbase and no preferred subbase classically entails $\neg\psi$. The last two entailment principles are not valid inference mechanisms in our framework, since the set of EXI (resp. ARG) consequences might be inconsistent.

4.7 Concluding Remarks

Past works on reasoning about inconsistency in AI have suffered from multiple flaws: (i) they apply to one logic at a time and are often invented for one logic after another. (ii) They assume that the AI researcher will legislate how applications resolve inconsistency even though the AI researcher may often know nothing about a specific application which may be built in a completely different time frame and geography than the AI researcher's work – in the real world, users are often stuck with the consequences of their decisions and would often like to decide what they want to do with their data (including what data to consider and what not to consider when there are inconsistencies). An AI system for reasoning about inconsistent information must support the user in his/her needs rather than forcing something down their throats. (iii) Most existing frameworks use some form or the other of maximal consistent subsets.

In this chapter, we attempt to address all these three flaws through a single unified approach that builds upon Tarksi's axiomatization of what a logic is. Most existing monotonic logics such as classical logic, Horn logic, probabilistic logic, temporal logic are special cases of Tarski's definition of a logic. Thus, we develop a framework for reasoning about inconsistency in any logic that satisfies Tarski's axioms. Second, we propose the notion of an "option" in any logic satisfying Tarski's axioms. An option is a set of formulas in the logic that is closed and consistent – however, the end user is not forced to choose a maximal consistent subset and options need not be maximal or even subsets of the original inconsistent knowledge base. Another element of our framework is that of preference. Users can specify any preference relation they want on their options.

Once the user has selected the logic he is working with, the options that he considers appropriate, and his preference relation on these options, our framework provides a semantics for a knowledge base *taking these user inputs into account*.

Our framework for reasoning about inconsistency has three basic components: (i) a set of options which are consistent and closed sets of formulas determined from the original knowledge base by means of a weakening mechanism which is general enough to apply to arbitrary logics and that allows users to flexibly specify how to weaken a knowledge base according to their application domains and needs. (ii) A general notion of preference relation between options. We show that our framework not only captures maximal consistent subsets, but also many other criteria that a user may use to select between options. We have also shown that by defining an appropriate preference relation over options, we can capture several existing works such as the subbases defined in [RM70] and Brewka's subtheories. (iii) The last component of the framework consists of an inference mechanism that allows the selection of the inferences to be drawn from the knowledge base. This mechanism should return an option. This forces the system to make safe inferences.

We have also shown through examples how this abstract framework can be used in different logics, provided new results on the complexity of reasoning about inconsistency in such logics, and proposed general algorithms for computing preferred options.

In short, our framework empowers end-users to make decisions about what they mean by an option, what options they prefer to what other options, and prevents them from being dragged down by some systemic assumptions made by a researcher who might never have seen their application or does not understand the data and/or the risks posed to the user in decision making based on some a priori definition of what data should be discarded when an inconsistency arises.

Chapter 5

PLINI: A Probabilistic Logic for Inconsistent News Information

The work described in this chapter appears in [AMB$^+$11].

5.1 Introduction and Motivating Example

Google alone tracks thousands news sites around the world on a continuous basis, collecting millions of news reports about a wide range of phenomena. While a large percentage of news reports are about different types of *events* (such as terrorist attacks, meetings of G-8 leaders, results of sporting events, to name a few), there are also other types of news reports such as editorials and style sections that may not always be linked to events, but to certain topics (which in turn may include events). For example, it is quite common to read editorials about a nuclear nonproliferation treaty or about a political candidate's attacks on his rival. Thus, even in news pieces that may not directly be about an event, there are often references to events.

In this chapter, we study the problem of *identifying inconsistency* in news reports about events. The need to reason about inconsistency is due to the fact that different news sources generate their individual stories about an event which may differ from one another. We do not try to develop methods to resolve the inconsistency or perform paraconsistent reasoning in this work. Existing methods for inconsistency resolution and paraconsistent logics [Bel77, BDP97, BS98, BS89, dC74, Fit91, FFP05, FFP07] can be used on top of what we propose.

For instance, we may have a single event (a bombing in Ahmedabad, India in July 2008) that generates the following different news reports.

(S1) *An obscure Indian Islamic militant group is claiming responsibility for a bombing attack that killed at least 45 people in a western Indian city.*[1]

(S2) *Police believe an e-mail claiming responsibility for the bombing that killed 45 people Saturday was sent from that computer in a Mumbai suburb.*[2]

(S3) *MUMBAI – Police carried out a manhunt here Tuesday, believing that the serial blasts that rocked the western Indian city of Ahmedabad over the weekend, killing 42 people, were hatched in a Mumbai suburb.*[3]

Any reader who reads these reports will immediately realize that, despite the inconsistencies, they all refer to the same event. The inconsistencies in the above reports fall into the categories below.

1. **Linguo-Numerical Inconsistencies.** (S1) says *at least* 45 people were killed; (S2) says 45 people were killed; (S3) says 42 people were killed. (S1) and (S3) as well as (S2) and (S3) are inconsistent.

[1] Canadian TV report on July 27, 2008.

[2] WBOC, based on an AP news report of July 28, 2008.

[3] The Wall Street Journal, based on an AP news report of July 30, 2008.

2. **Spatial Inconsistencies.** (S1) and (S3) are apparently (but not intuitively) inconsistent in terms of the geospatial location of the event. (S1) says the event occurred in a "western Indian city", while (S3) says the event occurred in Ahmedabad. An automated computational system may flag this as an inconsistency if it does not recognize that Ahmedabad is in fact a western Indian city.

3. **Temporal Inconsistencies.** (S2) says the bombing occurred on Saturday, while (S3) says the bombing occurred over the weekend. When analyzing when the event occurred, we need to realize that the "Saturday" in (S2) refers to the past Saturday, while the "weekend" referred to in (S3) is the past weekend. Without this realization - and the realization that Saturday is typically a part of a weekend, a system may flag this as inconsistent.

In fact, when reasoning about events, many other kinds of inconsistencies or apparent inconsistencies can also occur. For example, a report that says an event occurred within 5 miles of College Park, MD and another report that says the event occurred in Southwest DC would (intuitively) be mutually inconsistent. When reasoning about inconsistency in reporting about news events, we need to recognize several factors.

- Are two news reports referring to the same event or not? The answer to this question determines whether integrity constraints (e.g. ones that say that if two violent events are the same, then the number of victims should be the same) are applicable or not?

- Are the two event reports inconsistent or not? If the two events are deemed to be the same, then they should have "similar" attribute values. However, if the two events are considered to be different, then they may have dissimilar attribute values.

- A third problem, as mentioned above, is that inconsistency can arise in the linguistic terms used to describe news events. When should varying numbers, temporal references, and geospatial references be considered to be "close enough"? This plays an important role in determining whether news reports are inconsistent or not.

A problem arises because of circularity. The answer to the first question is based on whether the events in question have similar attribute values, while the answer to the second question says that equivalent events should have similar attribute values. The ability to distinguish whether two reports refer to the same event or not, and whether they are inconsistent or not, is key to the theory underlying PLINI. We start in Section 5.2 with an informal definition of what we mean by an event. In Section 5.3, we provide a formal syntax and semantics for PLINI-formulas that contain linguistically modified terms such as "about 5 miles from Ahmedabad", "over 50 people" and "the first weekend of May 2009." We briefly show how we can reason about linguistic modifications to numeric, temporal, and geospatial data. We discuss similarity functions in Section 5.4. Then, in Section 5.5, we provide a syntax for PLINI-programs. Section 5.6 provides a formal model theory and fixpoint semantics for PLINI-programs that is a variant of the semantics of generalized annotated programs [KS92]. The least fixpoint of the fixpoint operator associated with PLINI-programs allows us (with additional clustering algorithms) to infer that certain events should be considered identical, while other events should be considered different. This additional clustering algorithm is briefly described in Section 5.7. Finally, in Section 6.6, we describe our prototype implementation and experiments.

Figure 5.1 shows the architecture of our PLINI framework. We start with an information extraction program that extracts event information automatically from text sources. Our implementation uses T-REX [AS07], though other IE programs may be used as well. Information extracted from news sources is typically uncertain and may

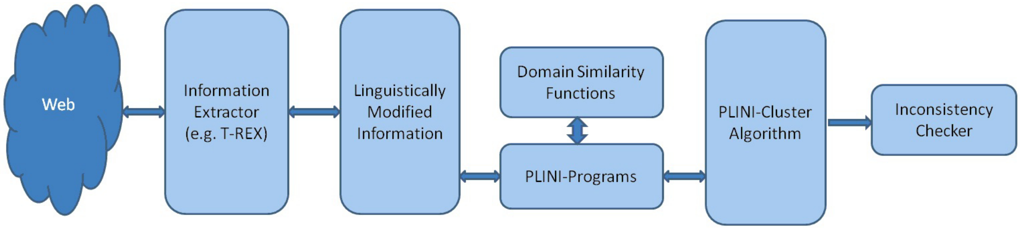

Web
Information Extractor (e.g. T-REX)
Linguistically Modified Information
Domain Similarity Functions
PLINI-Programs
PLINI-Cluster Algorithm
Inconsistency Checker

Figure 5.1: Architecture of the PLINI-system

include information that is linguistically modified such that from sentences **(S1), (S2), (S3)** above. Once the information extractor has identified events and extracted properties of those events, we need to identify which events are similar (and this in turn requires determining which properties of events are similar). To achieve this, we assume the existence of similarity functions on various data types – we propose several such functions for certain data types that are common in processing news information. PLINI-programs may be automatically extracted from training data using standard machine learning algorithms and a training corpus. The rules in a PLINI-program allow us to determine the similarity between different events. Our PLINI-Cluster algorithm clusters events together based on the similarity determined by the rules. All events within the same cluster are deemed equivalent. Once this is done, we can determine whether a real inconsistency exists or not.

Our experiments are based on event data extracted by the T-REX [AS07] system. T-REX has been running continuously for over three years. It primarily extracts information on violent events worldwide from over 400 news sources located in 130 countries. Over 126 million articles have been processed to date by T-REX which has automatically extracted a database of approximately 19 million property-value pairs related to violent events. We have conducted detailed experiments showing that the PLINI-architecture can identify inconsistencies with high precision and recall.

5.2 What is an Event?

We assume that every event has three kinds of properties: a spatial property describing the region where the event occurred, a temporal property describing the period of time when the event occurred, and a set of event-specific properties describing various aspects of the event itself. The event specific properties vary from one type of event to another. Some examples of events are the following.

- **Terrorist act:** Here, the spatial property describes the region where the event occurred (e.g. Mumbai suburb), and various event-specific properties such as number_of_victims, number_injured, weapon, claimed_responsibility, arrested, etc., can be defined.

- **Political meeting:** Here, the event specific properties might include attendee, photo, agreement_reached, etc.

- **Natural disaster:** The spatial properties in this case may be somewhat different from those above. For instance, if we consider the 2004 tsunami in the Indian ocean, the region where the event occurred may be defined as a set of regions (e.g. Aceh, Sri Lanka, and so forth), while the time scales may also be different based on when the tsunami hit the affected regions. The event-specific attributes might include properties such as number_of_victims, number_injured, number_houses_destroyed, property_damage_value, and so forth.

An event can be represented as a set of $(property, value)$ pairs. Table 5.1 describes the events presented in Section 5.1.

e_{S1}	$(type, "bombing\ attack"), (perpetrator, "Indian\ Islamic\ Militant\ Group"),$ $(place, "western\ Indian\ city"), (number\_of\_victims, "at\ least\ 45")$
e_{S2}	$(type, "bombing"), (date, "Saturday"),$ $(report\_time, 7/28/2008), (number\_of\_victims, 45)$
e_{S3}	$(type, "serial\ blast"), (number\_of\_victims, 42), (report\ time, 7/30/2008)$ $(place, "Ahmedabad"), (date, "over\ the\ weekend")$

Table 5.1: Examples of event descriptions

5.3 PLINI Wffs: Syntax and Semantics

As shown in Section 5.1, news reports contain statements that have numeric, spatial, and temporal indeterminacy. In this section, we introduce a multi-sorted logic syntax to capture such statements.

5.3.1 Syntax of Multi-sorted Wffs

Our definition of multi-sorted well formed formulas (mWFFs for short) builds upon well-known multi-sorted logics [RCC92] and modifies them appropriately to handle the kinds of linguistic modifiers used in news articles as exemplified in sentences (S1), (S2) and (S3). In this section, we introduce the syntax of mWFFs.

Throughout this chapter, we assume the existence of a set $\mathcal{S}$ of *sorts*. The set $\mathcal{S}$ includes sorts such as $Real$, $Time$, $Time\ Interval$, $Date$, $NumericInterval$, $Point$, $Space$, and $ConnectedPlace$. Each sort s has an associated set $dom(s)$ whose elements are called *constants* of sort s. For each sort $s \in \mathcal{S}$, we assume the existence of an infinite set $\mathcal{V}_s$ of variable symbols.

Definition 39 (Term). *A* term *t of sort s is any member of $dom(s) \cup \mathcal{V}_s$. A* ground *term is a constant.*

We assume the existence of a set $\mathcal{P}$ of predicate symbols. Each predicate symbol $p \in \mathcal{P}$ has an associated arity, $arity(p)$, and a *signature*. If a predicate symbol $p \in \mathcal{P}$ has arity n, then its signature is of the form $(s_1, \ldots, s_n)$ where each $s_i \in \mathcal{S}$ is a sort.

Definition 40 (Atom)**.** *If $p \in \mathcal{P}$ is a predicate symbol with signature $(s_1, \ldots, s_n)$, and $t_1, \ldots, t_n$ are (resp. ground) terms of sorts $s_1, \ldots, s_n$ respectively, then $p(t_1, \ldots, t_n)$ is a (resp. ground) atom.*

Definition 41 (mWFF)**.** *A* multi-sorted well formed formula (mWFF) *is defined as follows:*

- *Every atom is an mWFF (atomic mWFF).*
- *If A and B are mWFFs, then so are $A \wedge B$, $A \vee B$, and $\neg A$.*
- *If $s \in \mathcal{S}$, $X \in \mathcal{V}_s$, and A is an mWFF, then $\forall_s X.A$ and $\exists_s X.A$ are also mWFFs.*

We are now ready to give a semantics for the syntactic objects introduced above. We start with the definition of denotation of various syntactic constructs.

Definition 42 (Denotation)**.** *Suppose $s \in \mathcal{S}$ is a sort, and $c \in dom(s)$. Each sort s has a* fixed *associated* denotation universe *$\mathcal{U}_s$. Each ground term t of sort s and each predicate symbol $p \in \mathcal{P}$ has a* denotation *$[\![t]\!]$ ($[\![p]\!]$ resp.), defined as follows.*

- *$[\![c]\!]$ is an element of $\mathcal{U}_s$ for each $c \in dom(s)$.*
- *If $p \in \mathcal{P}$ is a predicate symbol with signature $(s_1, \ldots, s_n)$, then $[\![p]\!]$ is a subset of $\mathcal{U}_{s_1} \times \ldots \times \mathcal{U}_{s_n}$.*

This work considers the sorts: $Real$, $Time$, $Time\ Interval$, $Date$, $Point$, $Space$, and $ConnectedPlace$. We describe each of these sorts below.

$\mathbf{Real}$. $Real$ is a sort whose domain is the set $\mathbb{R}$ of real numbers. The denotation of symbols in $dom(Real)$ is:

- The denotation universe is $\mathcal{U}_{Real} = \mathbb{R}$.[4]
- For each symbol $r \in Real$, $[\![r]\!] = r \in \mathbb{R}$, i.e., real numbers denote themselves.

$\mathbf{Time}$. Let us assume that $Time$ is a sort having the set of symbols such as 2008, $08/2008$, $08/01/2008$, etc. as its domain.[5] The denotation of symbols in $dom(Time)$ can be defined as follows:

- The denotation universe is $\mathcal{U}_{Time} = \wp(\mathbb{Z})$ where $\mathbb{Z}$ is the set of non-negative integers and each $t \in \mathbb{Z}$ encodes a point in time, i.e. the number of time units elapsed since the origin of the time scale adopted by the user. As an example, $t \in \mathbb{Z}$ may encode the number of seconds elapsed since January 1^{st} 1970, 0:00:00 GMT.
- The denotation of each symbol $t' \in dom(Time)$ is an element of $\wp(\mathbb{Z})$, i.e. an unconstrained set of points in time.

$\mathbf{TimeInterval}$. $Time\ Interval$ is a sort whose domain is the set of symbols of the form $(start, end)$ where $start, end \in \mathbb{Z}$. The denotation of symbols in $dom(Time\ Interval)$ can be defined as follows:

- The denotation universe is $\mathcal{U}_{Time\ Interval} = \{I \in \wp(\mathbb{Z}) \mid I \text{ is connected}\}$.

[4] Though the domain and denotation universe of $Real$ are identical, this is not the case for all sorts (the sorts $Space$ and $ConnectedPlace$ below are examples).

[5] Formally, we could define this set of symbols as follows. Every non-negative integer is a *year*. Every integer from 1 to 12 is a month. Every integer from 1 to 31 is a day. Every year is in $dom(Time)$. If m is a month and y is a year, then m/y is in $dom(Time)$. If d is a day, m is a month, and y is a year, then $d/m/y$ is in $dom(Time)$. The fact that 31/2/2009 is not a valid date can be handled by adding an additional “validity” predicate. We do not go into this as this is not the point of this work.

- The denotation of each symbol $(\mathit{start}, \mathit{end}) \in \mathit{dom}(\mathit{Time\ Interval})$ is defined in the obvious manner: $[\![(\mathit{start}, \mathit{end})]\!] = [\mathit{start}, \mathit{end})$ — note that this is a left-closed, right open interval.

Date. Let us assume that Date is a sort having the set of symbols of the form month-day-year as its domain and $\mathit{dom}(\mathit{Date}) \subset \mathit{dom}(\mathit{Time\ Interval})$. The denotation of symbols in $\mathit{dom}(\mathit{Date})$ can be defined as follows:

- The denotation universe is $\mathcal{U}_{\mathit{Date}} = \{D \in \mathcal{U}_{\mathit{Time\ Interval}} \mid \mathit{sup}(D) - \mathit{inf}(D) = \tau \wedge \mathit{inf}(D) \bmod \tau = 0\}$, where τ is the number of time units, in the selected time scale, contained in a day. For example, if the adopted time scale has a granularity of hours, then $\tau = 24$.

Point. Point is a sort whose domain is the set $\mathbb{R} \times \mathbb{R}$. The denotation of symbols in $\mathit{dom}(\mathit{Point})$ can be defined as follows:

- The denotation universe is $\mathcal{U}_{\mathit{Point}} = \mathbb{R} \times \mathbb{R}$.

- For each symbol $p = (r_1, r_2) \in \mathit{dom}(\mathit{Point})$, $[\![p]\!]$ is the point $p = (r_1, r_2) \in \mathbb{R} \times \mathbb{R}$.

Space. Space is a sort whose domain is an enumerated set of strings such as $\mathit{Atlantic\ Ocean}$, $\mathit{Great\ Lakes}$, $WashingtonDC$, etc. The denotation of symbols in $\mathit{dom}(\mathit{Space})$ can be defined as follows:

- The denotation universe is $\mathcal{U}_{\mathit{Space}} = \wp(\mathbb{R} \times \mathbb{R})$, where $\wp(\mathbb{R} \times \mathbb{R})$ is the power set of $\mathbb{R} \times \mathbb{R}$.

- For each symbol $a \in \mathit{dom}(\mathit{Space})$, $[\![a]\!]$ is a member of $\wp(\mathbb{R} \times \mathbb{R})$, i.e. an unconstrained set of points in $\mathbb{R} \times \mathbb{R}$.

For instance, the denotation, $[\![Paris]\!]$, of *Paris*, is a set of points on the 2-dimensional Cartesian plane that corresponds to the region referred to as *Paris*. Another example of an element of sort *Space* is *United States*, whose denotation is the set of points that is the union of all points in the real plane corresponding to each of the regions that form the country (continental US, Alaska, Hawaii, etc.).

Connected Place. *ConnectedPlace*'s domain is the subset of *Space*'s domain that consists of connected regions. The denotation of symbols in $dom(\mathit{ConnectedPlace})$ can be defined as follows:

- The denotation universe is $\mathcal{U}_{\mathit{ConnectedPlace}} = \{a \in \mathcal{U}_{\mathit{Space}} \mid a \text{ is connected}\}$.
- For each symbol $l \in dom(\mathit{ConnectedPlace})$, $[\![l]\!]$ is a connected element l of $\wp(\mathbb{R} \times \mathbb{R})$, i.e. a connected region in $\mathbb{R} \times \mathbb{R}$ that corresponds to l. Thus, $[\![\mathit{Washington\ DC}]\!]$ might be the set $\{(x, y) \mid 10 \leq x \leq 12 \wedge 36 \leq y \leq 40\}$ and $[\![Paris]\!]$ might be similarly defined.

Note that while *continental US* is an element of sort *ConnectedPlace*, *United States* is not because the US is not a connected region. ***Throughout the rest of this chapter we assume an arbitrary but fixed denotation function*** $[\![.]\!]$ ***for each constant and predicate symbol in our language.***

Definition 43 (Assignment)**.** *An assignment* σ *is a mapping,* $\sigma : \cup_{s \in \mathcal{S}} \mathcal{V}_s \to \cup_{s \in \mathcal{S}} \mathcal{U}_s$ *such that for every* $X \in \mathcal{V}_s$*,* $\sigma(X) \in \mathcal{U}_s$*.*

Thus σ assigns an element of the proper sort for every variable. We write $\sigma[A]$ to denote the simultaneous replacement of each variable X in A by $\sigma(X)$.

Definition 44 (Semantics of mWFFs)**.** *The evaluation of an mWFF under assignment* σ *is defined as follows:*

Predicate Symbol	***Signature***	***Denotation***	***Associated Region***
$almost$	$(Real, Real, Real)$	$\{(x, \epsilon, y) \mid x, \epsilon, y \in \mathbb{R}$ $(0 < [\![\epsilon]\!] \leq 1) \ \wedge \ ((1 - \epsilon) \times x \leq y < x)\}$	The interval $[(1 - \epsilon) \times x, x)$
at_least	$(Real, Real, Real)$	$\{(x, \epsilon, y) \mid x, \epsilon, y \in \mathbb{R}$ $(0 \leq [\![\epsilon]\!] \leq 1) \ \wedge \ (x \leq y \leq (x + (x \times \epsilon))\}$	The interval $[x, x + (\epsilon \times x)]$
$around$	$(Real, Real, Real)$	$\{(x, \epsilon, y) \mid x, \epsilon, y \in \mathbb{R} \ \wedge \ (0 \leq [\![\epsilon]\!] \leq 1)$ $(x - (x \times \epsilon) \leq y \leq x + (x \times \epsilon))\}$	The interval $[x - (\epsilon \times x), x + (\epsilon \times x)]$
$most\_of$	$(Real, Real, Real)$	$\{(x, \epsilon, y) \mid x, \epsilon, y \in \mathbb{R} \ \wedge \ (0.0 < [\![\epsilon]\!] < 0.5)$ $(x \times (1 - \epsilon) \leq y < x)\}$	The interval $[x - (x \times \epsilon), x)$
$between$	$(Real, Real, Real, Real)$	$\{(x, y, \epsilon, z) \mid x, y, z, \epsilon \in \mathbb{R} \ \wedge (0 \leq [\![\epsilon]\!] \leq 1)$ $(x - (x \times \epsilon) \leq z \leq y + (y \times \epsilon))\}$	The interval $[x - (x \times \epsilon), y + (y \times \epsilon)]$

Table 5.2: Denotations for selected linguistically modified numeric predicates

1. *If p is a predicate symbol of arity n and signature $(s_1, \ldots, s_n)$, and $t_1, \ldots, t_n$ are terms of sort $s_1, \ldots, s_n$ respectively, then the atomic mWFF $\sigma[p(t_1, \ldots, t_n)]$ is true* iff $([\![\sigma(t_1)]\!], \ldots, [\![\sigma(t_n)]\!]) \in [\![p]\!]$.

2. *If A is an mWFF, then $\sigma[\neg A]$ is true* iff *$\sigma[A]$ is not true.*

3. *If A and B are both mWFFs, then $\sigma[A \wedge B]$ is true iff $\sigma[A]$ is true and $\sigma[B]$ is true.*

4. *If A and B are both mWFFs, then $\sigma[A \vee B]$ is true iff $\sigma[A]$ is true or $\sigma[B]$ is true.*

5. *If A is an mWFF and $X \in \mathcal{V}_s$, then $\sigma[\forall_s X.A]$ is true* iff *for each possible assignment τ, identical to σ except possibly for X, $\tau[A]$ is true.*

6. *If A is an mWFF and $X \in \mathcal{V}_s$, then $\sigma[\exists X.A]$ is true* iff *there is an assignment τ, identical to σ except possibly for X, for which $\tau[A]$ is true.*

An mWFF A is true iff $\sigma[A]$ is true for all assignments σ.

The above definitions describe the syntax and semantics of mWFFs. It should be clear from the preceding examples that we can use the syntax of mWFFs to reason about numbers with attached linguistic modifiers (e.g. "around 25", "between 25 and 30". "at least 40"), about time with linguistic modifiers (e.g. "last month", "morning of June 1, 2009",) and spatial information with linguistic modifiers (e.g. "center of Washington DC", "southwest of Washington DC").

Predicate Symbol	***Signature***	***Denotation***	***Associated Region***
Positional Indeterminacy			
$center$	$(ConnectedPlace, Real, Point)$	$\{(l, \delta, p) \mid l \in \mathcal{U}_{ConnectedPlace} \ \wedge \ \delta \in [0, 1]$ $\wedge \ p \in \mathcal{U}_{Point} \ \wedge \ d(p, Cent(l)) \leq \delta \cdot hside(l)\}$	Circle centered at the center of the rectangle maximally contained in l^6, with radius equal to a fraction δ of half the length of the smaller side of the rectangle
$boundary$	$(Space, Point)$	$\{(a, p) \mid a \in \mathcal{U}_{Space} \ \wedge \ p \in \mathcal{U}_{Point}$ $\wedge \ (\forall \epsilon > 0 : (\exists p_1 \in a, \ p_2 \notin a : d(p_1, p) < \epsilon$ $\wedge \ d(p_2, p) < \epsilon))\}$	Points on the edge of a
Distance Indeterminacy			
$distance$	$(Space, Real, Point)$	$\{(a, r, p) \mid a \in \mathcal{U}_{Space} \ \wedge \ r \in \mathbb{R} \ \wedge \ p \in \mathcal{U}_{Point}$ $\wedge \ (\exists p_0 \in a : d(p_0, p) = r)\}$	Points at a distance r from a point in a
$within$	$(Space, Real, Point)$	$\{(a, r, p) \mid a \in \mathcal{U}_{Space} \ \wedge \ r \in \mathbb{R} \ \wedge \ p \in \mathcal{U}_{Point}$ $\wedge \ (\exists p_0 \in a : d(p_0, p) \leq r)\}$	Points at a distance r or less from a point in a
Directional Indeterminacy			
$north$	$(Space, Real, Space)$	$\{(a, \theta, p) \mid a \in \mathcal{U}_{Space} \ \wedge \ \theta \in \mathbb{R} \ \wedge \ p \in \mathcal{U}_{Point}$ $\wedge \ (\exists p_0 \in a : p \in NCone(\theta, p_0))\}$	$NCone(\theta, p)$: ℓ_0 upwards parallel to the Y-axis
ne	$(Space, Real, Space)$	$\{(a, \theta, p) \mid a \in \mathcal{U}_{Space} \ \wedge \ \theta \in \mathbb{R} \ \wedge \ p \in \mathcal{U}_{Point}$ $\wedge \ (\exists p_0 \in a : p \in NECone(\theta, p_0))\}$	$NECone(\theta, p)$: ℓ_0 to the right with slope 1
nw	$(Space, Real, Space)$	$\{(a, \theta, p) \mid a \in \mathcal{U}_{Space} \ \wedge \ \theta \in \mathbb{R} \ \wedge \ p \in \mathcal{U}_{Point}$ $\wedge \ (\exists p_0 \in a : p \in NWCone(\theta, p_0))\}$	$NWCone(\theta, p)$: ℓ_0 to the left with slope -1
$south$	$(Space, Real, Space)$	$\{(a, \theta, p) \mid a \in \mathcal{U}_{Space} \ \wedge \ \theta \in \mathbb{R} \ \wedge \ p \in \mathcal{U}_{Point}$ $\wedge \ (\exists p_0 \in a : p \in SCone(\theta, p_0))\}$	$SCone(\theta, p)$: ℓ_0 downwards parallel to the Y-axis

Table 5.3: Denotations for selected linguistically modified spatial predicates

Table 5.2 shows denotations of some predicate symbols for linguistically modified numbers, while and Tables 5.3 and 5.4 do the same for linguistically modified geospatial and temporal quantities, respectively.

Example 38 (Semantics for linguistically modified numbers)**.** *Consider the predicate symbol $most\_of$ in Table 5.2. Given $0 < \epsilon < 0.5$, we say that $most\_of(x, \epsilon, y)$ is true (y is "most of" x) iff $x \times (1 - e) \leq y \leq x$. Thus, when $x = 4, e = 0.3, y = 3.1$, we see that y lies between 2.8 and 4 and hence $most\_of(4, 0.3, 3.1)$ holds. However, if $e = 0.2$, then $most\_of(4, 0.2, 3.1)$ does not hold because y must lie in the interval* $[3.2, 4]$.

Example 39 (Semantics for linguistically modified spatial concepts)**.** *Consider the predicate symbol $boundary$ defined in Table 5.3 ($boundary$ is defined with respect to a set of points in a 2-dimensional space) and consider the rectangle a' defined by the constraints $1 \leq x \leq 4$ and $1 \leq y \leq 5$. A point p is on the boundary of a iff for all $\epsilon > 0$, there is a point $p_1 \in a$ and a point $p_2 \notin a$ such that the distance between p and each of p_1, p_2 is less*

[6] We are assuming there is one such rectangle; otherwise a more complex method is used.

Predicate Symbol	***Signature***	***Denotation***	**Associated Region**
$morning$	$(Date, Date)$	$\{(d_1, d_2) \mid d_1, d_2 \in \mathcal{U}_{Date} \wedge$ $GLB(d_1) \leq d_2 \leq (GLB(d_1) + LUB(d_1)/2\}$	The entire first half of a day

$last\_month$	$(Date, Date)$	for $m = 1$, $\{((m, d_0, y), z) \mid (m, d_0, y), z \in \mathcal{U}_{Date} \wedge$
		$(\exists i)\ s.t.\ (12, i, y-1) \in Date\ \wedge\ z \in [\![(12, i, y-1)]\!]\}$
		for $m \geq 2$, $\{((m, d_0, y), z) \mid (m, d_0, y) \in \mathcal{U}_{Date}, z \in \mathcal{U}_{Time}$
		$\wedge\ (\exists i)\ s.t.\ (m-1, i, y) \in Date\ \wedge\ z \in [\![(m-1, i, y)]\!]\}$
			The denotation of the month immediately preceding m

$around$	$(Date, Real, Time\ Interval)$	$\{((m, d_0, y), k, (z_s, z_e)) \mid (m, d_0, y) \in \mathcal{U}_{Date}$ $\wedge\ z_s, z_e \in \mathcal{U}_{Time}\ \wedge\ k \in Real\ \wedge\ z_s = inf((m_s, d_s, y_s)\ \wedge$ $z_e = sup((m_e, d_e, y_e))\}$, where (m_s, d_s, y_s) and (m_e, d_e, y_e) refer to the days which are exactly k days before and after (m, d_0, y)	The time points which are within a few days of a given date
$shortly\_before$	$(Date, Real, Time\ Interval)$	$\{((m, d_0, y), k, (z_s, z_e)) \mid (m, d_0, y) \in \mathcal{U}_{Date}$ $\wedge\ z_s, z_e \in \mathcal{U}_{Time}\ \wedge\ k \in \mathcal{U}_{Real}\ \wedge\ z_s = inf((m_s, d_s, y_s))$ $\wedge\ z_e = inf((m, d_0, y))]\}$, where (m_s, d_s, y_s) refers to the day which is exactly k days before (m, d_0, y)	The period shortly before a given date
$shortly\_after$	$(Date, Real, Time\ Interval)$	$\{((m, d_0, y), k, (z_s, z_e)) \mid (m, d_0, y) \in \mathcal{U}_{Date}$ $\wedge\ z_s, z_e \in \mathcal{U}_{Time}\ \wedge\ k \in \mathcal{U}_{Real}\ \wedge\ z_s = sup((m, d_0, y))$ $\wedge\ z_e = inf((m_e, d_e, y_e))]\}$, where (m_e, d_e, y_e) refers to the day which is exactly k days after (m, d_0, y)	The period shortly after a given date

Table 5.4: Denotations for selected linguistically modified temporal predicates

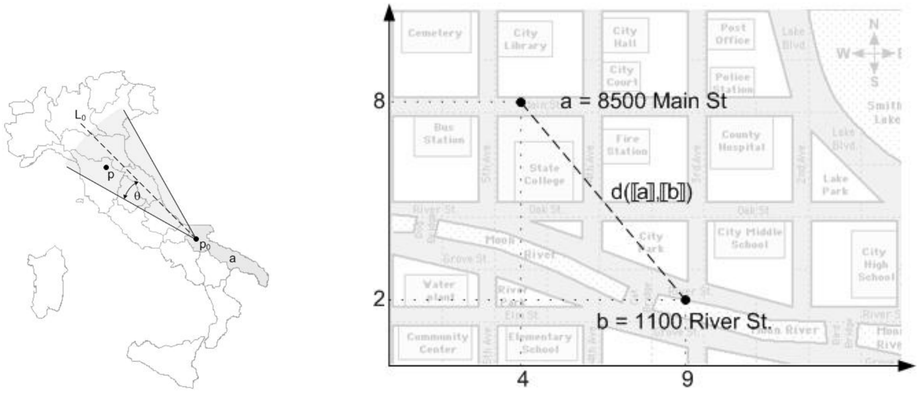
than ϵ. Using this definition, we see immediately that the point $(1,1)$ is on the boundary of the rectangle a', but $(1,2)$ is not.

Now consider the predicate symbol nw defining the northwest of a region (set of points). According to this definition, a point p is to the northwest of a region a w.r.t. cone-angle θ iff there exists a point p_0 in a such that p is in $NWCone(\theta, p_0)$. $NWCone(\theta, p_0)$[7] is defined to be the set of all points p' obtained by (i) drawing a ray L_0 of slope -1 to the left of vertex p_0, (ii) drawing two rays with vertex p_0 at an angle of $\pm\theta$ from L_0 and (iii) looking at between the two rays in item (ii). Figure 5.2(a) shows this situation. Suppose a is the shaded region and $\theta = 20$ (degrees). We see that p is to the northwest of this region according to the definition in Table 5.3.

5.4 Similarity Functions

We now propose similarity functions for many of the major sorts discussed in this chapter. We do not claim that these are the only definitions – many definitions are possible, often based on application needs. We merely provide a few in order to illustrate that reasonable definitions of this kind exist.

[7] The other cones referenced in Table 5.3 can be similarly defined.



(a) (b)

Figure 5.2: Example of (a) point p in the northwest of a region a; (b) application of sim_1^P and sim_2^P

We assume the existence of an arbitrary but fixed denotation function for each sort. Given a sort s, a similarity function is a function $sim^s : dom(s) \times dom(s) \to [0, 1]$, which assigns a degree of similarity to each pair of elements in $dom(s)$. All similarity functions are required to satisfy two very basic axioms.

$$sim^s(a, a) = 1 \tag{5.1}$$

$$sim^s(a, b) = sim^s(b, a) \tag{5.2}$$

Sort *Point*

Consider the sort $Point$, with denotation universe $\mathcal{U}_{Point} = \mathbb{R} \times \mathbb{R}$. Given two terms a and b of sort $Point$, we can define the similarity between a and b in any of the following ways.

$$sim_1^P(a,b) = e^{-\alpha \cdot d([\![a]\!],[\![b]\!])} \tag{5.3}$$

where $d([\![a]\!],[\![b]\!])$ is the distance in $\mathbb{R} \times \mathbb{R}$ between the denotations of a and b[8], and α is a factor that controls how fast the similarity decreases as the distance increases.

$$sim_2^P(a,b) = \frac{1}{1 + \alpha \cdot d([\![a]\!],[\![b]\!])} \tag{5.4}$$

where $d()$ and α have the same meaning as in Equation 5.3.

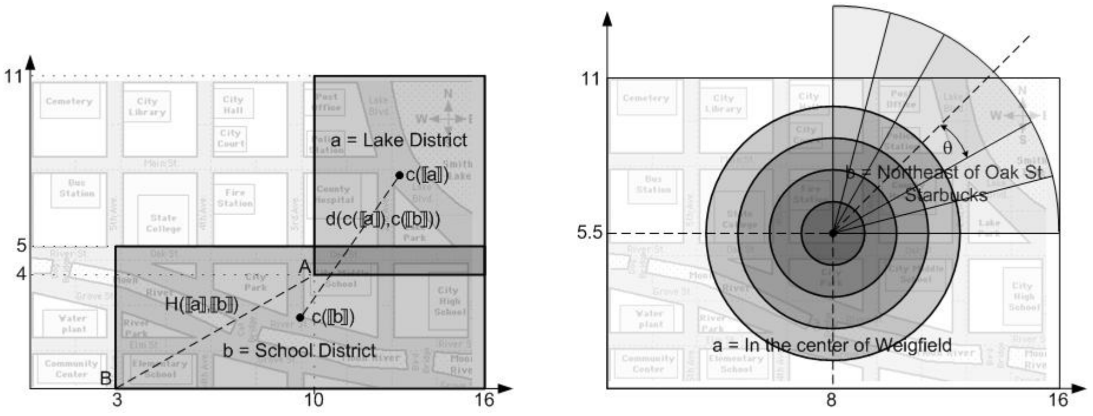
Example 40. *Assuming that street addresses can be approximated as points, consider the points* $a =$ *"8500 Main St." and* $b =$ *"1100 River St." in Figure 5.2(b), with denotations* $(4,8)$ *and* $(9,2)$ *respectively. Assuming* $\alpha = 0.3$*, then* $d([\![a]\!],[\![b]\!]) = 7.81$*,* $sim_1^P(a,b) = 0.096$*, and* $sim_2^P(a,b) = 0.299$.

Sort *ConnectedPlace*

Consider the sort $ConnectedPlace$, with denotation universe $\mathcal{U}_{ConnectedPlace} = \{a \in \mathcal{U}_{Space} \mid a \text{ is connected}\}$. Given two terms a and b of sort $ConnectedPlace$, the similarity between a and b can be defined in any of the following ways.

$$sim_1^{CP}(a,b) = e^{-\alpha \cdot d(c([\![a]\!]),c([\![b]\!]))} \tag{5.5}$$

[8] If elements in $\mathcal{U}_{Point}$ are pairs of latitude, longitude coordinates, then $d()$ is the great-circle distance. We will assume that $d()$ is the Euclidean distance, unless otherwise specified.



(a) (b)

Figure 5.3: Example of the application of similarity functions for sort $ConnectedPlace$

where $c(\llbracket a \rrbracket)$, $c(\llbracket b \rrbracket)$ in $\mathbb{R} \times \mathbb{R}$ are the centers of $\llbracket a \rrbracket$ and $\llbracket b \rrbracket$ respectively, $d(c(\llbracket a \rrbracket), c(\llbracket b \rrbracket))$ is the distance between them, and α is a factor that controls how fast the similarity decreases as the distance between the centers of the two places increases. This similarity function works well when comparing geographic entities at the same level of granularity. When places can be approximated with points, it is equivalent to $sim_1^P(a, b)$.

$$sim_2^{CP}(a, b) = \frac{1}{1 + \alpha \cdot d(c(\llbracket a \rrbracket), c(\llbracket b \rrbracket))} \tag{5.6}$$

where $c()$, $d()$ and α have the same meaning as in Equation 5.5.

Example 41. *Consider the two places* $a =$ *"Lake District" and* $b =$ *"School District" in Figure 5.3(a), and suppose their denotations are the two shaded rectangles in the figure. It is easy to observe that* $c(\llbracket a \rrbracket) = (13, 7.5)$*,* $c(\llbracket b \rrbracket) = (9.5, 2.5)$*, and* $d(c(\llbracket a \rrbracket), c(\llbracket b \rrbracket)) = 6.103$*. Hence, for* $\alpha = 0.3$*,* $sim_1^{CP}(a, b) = 0.160$ *and* $sim_2^{CP}(a, b) = 0.353$*.*

Other two similarity functions can be defined in terms of the areas of the two regions.

$$sim_3^{CP}(a, b) = \frac{A(\llbracket a \rrbracket \cap \llbracket b \rrbracket)}{A(\llbracket a \rrbracket \cup \llbracket b \rrbracket)} \tag{5.7}$$

where $A(\llbracket t \rrbracket)$ is a function that returns the area of $\llbracket t \rrbracket$. Intuitively, this function uses the amount of overlap between the denotations of a and b as their similarity.

$$sim_4^{CP}(a,b) = \frac{A(\llbracket a \rrbracket \cap \llbracket b \rrbracket)}{\max_{t \in \{a,b\}} A(\llbracket t \rrbracket)} \tag{5.8}$$

where $A(\llbracket t \rrbracket)$ has the same meaning as in Equation 5.7.

Example 42. *Consider the two connected places $a =$ and $b =$ in Figure 5.3(a), and their respective denotations. The intersection of the two denotations is the darker shaded region, whereas their union is the whole shaded region. It is straightforward to see that $A(\llbracket a \rrbracket) = 42$, $A(\llbracket b \rrbracket) = 65$, $A(\llbracket a \rrbracket \cap \llbracket b \rrbracket) = 6$, and $A(\llbracket a \rrbracket \cup \llbracket b \rrbracket) = 101$. Thus, $sim_3^{CP}(a,b) = 0.059$ and $sim_4^{CP}(a,b) = 0.092$*

In order to better illustrate the great expressive power of our framework, we now consider a more complex scenario, where the terms being compared are linguistically modified terms. We show how the similarity of such terms depends on the specific denotations assumed by the user for each predicate symbol.

Example 43. *Consider the two linguistically modified terms of sort $ConnectedPlace$ $a =$ "In the center of Weigfield" and $b =$ "Northeast of Oak St. Starbucks", where Weigfield is the fictional city depicted in Figure 5.3. Assuming the denotation of $center$ and ne shown in Table 5.3, we now compute the similarity between a and b for different values of δ and θ. Figure 5.3(b) shows denotations of a for values of δ of 0.2, 0.4, 0.6, and 0.8, and denotations of b for values of θ of $15°$, $30°$, and $45°$. In order to simplify similarity computation, we make the following assumptions (without loss of generality): (i) the term "Oak St. Starbucks" can be interpreted as a term of sort $Point$; (ii) the denotation of "Oak St. Starbucks" coincides with the geometrical center $(8, 5.5)$ of the bounding box of $\llbracket$"Weigfield"$\rrbracket$; (iii) the cones do not extend indefinitely, but rather within a fixed radius*

	$\delta = 0.2$	$\delta = 0.4$	$\delta = 0.6$	$\delta = 0.8$
$\theta = 15°$	0.0132	0.0276	0.0346	0.0380
$\theta = 30°$	0.0157	0.0413	0.0593	0.0699
$\theta = 45°$	0.0167	0.0494	0.0777	0.0970

Table 5.5: Value of $sim_3^{CP}(a, b)$ for different values of δ and θ

(8 units in this example) from their vertex. Table 5.5 reports the value of $sim_3^{CP}(a, b)$ for different values of δ and θ. The highest similarity corresponds to the case where $\delta = 0.8$ and $\theta = 45°$, which maximizes the overlap between the two regions. Intuitively, this result tells us that a user with a very restrictive interpretation of $center$ and ne (i.e., $\delta \ll 1$ and $\theta \ll 90°$ respectively) will consider a and b less similar than a user with a more relaxed interpretation of the same predicates.

Another similarity function can be defined in terms of the Hausdorff distance [Mun74].

$$sim_5^{CP}(a, b) = e^{-\alpha \cdot H([\![a]\!], [\![b]\!])} \tag{5.9}$$

where $H(P, Q) = \max(h(P, Q), h(Q, P))$, with $P, Q \in \wp(\mathbb{R} \times \mathbb{R})$, is the Hausdorff distance, where $h(P, Q) = \max_{p \in P} \ min_{q \in Q} d(p, q)$ is the distance between the point $p \in P$ that is farthest from any point in Q and the point $q \in Q$ that is closest to p. Intuitively, the Hausdorff distance is a measure of the mismatch between P and Q; if the Hausdorff distance is d, then every point of P is within distance d of some point of Q and vice versa.

Example 44. *Consider again the two connected places a = "Lake District" and b = "School District" in Figure 5.3, and their respective denotations. In this example, the Hausdorff distance between $[\![a]\!]$ and $[\![b]\!]$ can be interpreted as the distance between the two points A and B shown in the figure. Therefore, $H([\![a]\!], [\![b]\!]) = 8.062$ and $sim_5^{CP}(a, b) =$*

0.089 *for* $\alpha = 0.3$*. Exchanging the roles of* $[\![a]\!]$ *and* $[\![b]\!]$ *would lead to a shorter value of the distance, whereas H() selects the maximum.*

$$sim_6^{CP}(a,b) = e^{-\alpha \cdot d(c([\![a]\!]),c([\![b]\!]))} \cdot e^{-\beta \cdot (1-o([\![a]\!],[\![b]\!]))} \tag{5.10}$$

where $c()$, $d()$ and α have the same meaning as in Equation 5.5, $o([\![a]\!],[\![b]\!]) = \frac{A([\![a]\!] \cap [\![b]\!])}{A([\![a]\!] \cup [\![b]\!])}$ is the amount of overlap between $[\![a]\!]$ and $[\![b]\!]$, and β is a factor that controls how fast the similarity decreases as the amount of overlap between the two places decreases[9].

Example 45. *Consider again the two connected places in Figure 5.3, and their respective denotations. In this example,* $sim_6^{CP}(a,b) = 0.056$ *for* $\alpha = 0.3$ *and* $\beta = 0.5$.

The similarity function $sim_{1'}^{CP}$ below considers two places equivalent when their denotations are included into one another. We can define $sim_{2'}^{CP}, \ldots, sim_{6'}^{CP}$ in a similar way by modifying $sim_2^{CP}, \ldots, sim_6^{CP}$ analogously.

$$sim_{1'}^{CP}(a,b) = \begin{cases} 1 \text{ if } [\![a]\!] \subseteq [\![b]\!] \vee [\![b]\!] \subseteq [\![a]\!] \\ sim_1^{CP}(a,b) \text{ otherwise} \end{cases} \tag{5.11}$$

Sort *Space*

Consider the sort *Space*, with denotation universe $\mathcal{U}_{Space} = \wp(\mathbb{R} \times \mathbb{R})$, where $\wp(\mathbb{R} \times \mathbb{R})$ is the power set of $\mathbb{R} \times \mathbb{R}$. Given a term a of sort *Space*, let $P([\![a]\!])$ denote a subset of $\mathcal{U}_{ConnectedPlace}$ such that $\bigcup_{x \in P([\![a]\!])} x = [\![a]\!]$, elements in $P([\![a]\!])$ are pairwise disjoint and maximal, i.e. $\nexists y \in \mathcal{U}_{ConnectedPlace}, x_1, x_2 \in P([\![a]\!])\ s.t.\ y = x_1 \cup x_2$. Intuitively, $P([\![a]\!])$ is the set of the denotations of all the connected components of a. Given two terms a and

[9] Alternatively, one could specify $o([\![a]\!],[\![b]\!]) = \frac{A([\![a]\!] \cap [\![b]\!])}{\max_{t \in \{a,b\}} A([\![t]\!])}$

b of sort $Space$, the distance between a and b may be defined in many ways – two are shown below.

$$d_c^S(a,b) = \operatorname*{avg}_{a_i \in P(\llbracket a \rrbracket), b_i \in P(\llbracket b \rrbracket)} d(c(a_i), c(b_i)) \tag{5.12}$$

where $c()$ and $d()$ have the same meaning as in Equation 5.5.

$$d_h^S(a,b) = \operatorname*{avg}_{a_i \in P(\llbracket a \rrbracket), b_i \in P(\llbracket b \rrbracket)} H(a_i, b_i) \tag{5.13}$$

where $H()$ is the Hausdorff distance.

Intuitively d_c^S and d_h^S measure the average distance between any two connected components of the two spaces being compared. Alternatively, the avg operator could be replaced by either $\min$ or $\max$. As in the case of sort $ConnectedPlace$, a similarity function over sort $Space$ can be defined in any of the following ways.

$$sim_1^S(a,b) = e^{-\alpha \cdot d_c^S(a,b)} \tag{5.14}$$

$$sim_2^S(a,b) = \frac{1}{1+\alpha \cdot d_c^S(a,b)} \tag{5.15}$$

where d_c^S is the distance defined by Equation 5.12 and α is a factor that controls how fast the similarity decreases as the distance increases.

Example 46. *Consider the terms $a =$ "City buildings" and $b =$ "Schools" of sort $Space$ in Figure 5.4 with denotations $\llbracket a \rrbracket = \{a_1, a_2\}$ and $\llbracket b \rrbracket = \{b_1, b_2\}$ respectively. By computing and averaging the distances between the centers of all pairs $a_i, b_j \in P(\llbracket a \rrbracket) \times P(\llbracket b \rrbracket)$*

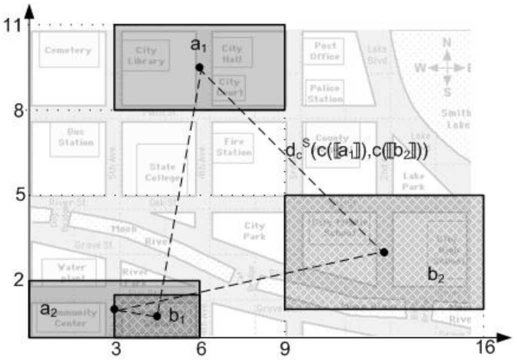


Figure 5.4: Example of application of similarity functions for sort $Space$

(see dashed lines in the figure), we obtain $d_c^S(a,b) = 7.325$ *and* $sim_1^S(a,b) = 0.111$*, and* $sim_2^S(a,b) = 0.313$ *for* $\alpha = 0.3$*.*

$$sim_3^S(a,b) = \frac{A(\llbracket a \rrbracket \cap \llbracket b \rrbracket)}{A(\llbracket a \rrbracket \cup \llbracket b \rrbracket)} \tag{5.16}$$

$$sim_4^S(a,b) = \frac{A(\llbracket a \rrbracket \cap \llbracket b \rrbracket)}{\max_{t \in \{a,b\}} A(\llbracket t \rrbracket)} \tag{5.17}$$

where $A(\llbracket t \rrbracket)$ is a function that returns the area of $\llbracket t \rrbracket$.

$$sim_5^S(a,b) = e^{-\alpha \cdot d_h^S(a,b)} \tag{5.18}$$

where d_h^S is the distance defined by Equation 5.13 and α is a factor that controls how fast the similarity decreases as the distance increases.

$$sim_6^S(a,b) = e^{-\alpha \cdot d_c^S(a,b)} \cdot e^{-\beta \cdot (1 - o(\llbracket a \rrbracket, \llbracket b \rrbracket))} \tag{5.19}$$

where d_c^S is the distance defined by Equation 5.12, α has the usual meaning, $o([\![a]\!], [\![b]\!]) = \frac{A([\![a]\!] \cap [\![b]\!])}{A([\![a]\!] \cup [\![b]\!])}$ is the amount of overlap between $[\![a]\!]$ and $[\![b]\!]$, and β is a factor that controls how fast the similarity decreases as the overlap between the two places decreases.

Example 47. *Consider again the two terms of sort $Space$ in Figure 5.4. It is straightforward to see that $A([\![a]\!]) = 30$, $A([\![b]\!]) = 32.5$, $A([\![a]\!] \cap [\![b]\!]) = 4.5$, and $A([\![a]\!] \cup [\![b]\!]) = 58$. Therefore, $sim_3^S(a, b) = 0.078$, $sim_4^S(a, b) = 0.138$, and $sim_6^S(a, b) = 0.044$, for $\alpha = 0.3$ and $\beta = 1$.*

Sort $Time\ Interval$

Consider the sort $Time\ Interval$, with denotation universe $\mathcal{U}_{Time\ Interval} = \{I \in \wp(\mathbb{Z}) \mid I \text{ is connected}\}$[10]. Given two terms a and b of sort $Time\ Interval$, the similarity between a and b can be defined in any of the following ways.

$$sim_1^{TI}(a, b) = e^{-\alpha \cdot |c([\![a]\!]) - c([\![b]\!])|} \tag{5.20}$$

where, for each time interval $t \in dom(Time\ Interval)$, $c([\![t]\!]) = \text{avg}_{z \in [\![t]\!]} z$ is the center of $[\![t]\!]$, and α is a factor that controls how fast the similarity decreases as the distance between the centers of the two time intervals increases.

$$sim_2^{TI}(a, b) = \frac{1}{1 + \alpha \cdot |c([\![a]\!]) - c([\![b]\!])|} \tag{5.21}$$

where $c()$ and α have the same meaning as in Equation 5.20.

Example 48. *Consider the two terms of sort $Time\ Interval$ a = "around May 13, 2009" and b = "shortly before May 16, 2009", and assume that the denotation of $around$ is a*

[10] Each $t \in \mathbb{Z}$ encodes a point in time, i.e. the number of time units elapsed since the origin of the time scale adopted by the user.

time interval extending 4 *days before and after the indeterminate date, and the denotation of* $shortly\_before$ *is the time interval extending* 2 *days before the indeterminate date. Then,* $[\![a]\!]$ *is the time interval* $[05/9/09, 05/17/09]$ *and* $[\![b]\!]$ *is the time interval* $[05/14/09, 05/16/09]$. *Assuming a time granularity of days, we have* $c([\![a]\!]) = 05/13/09$ *and* $c([\![b]\!]) = 05/15/09$[11]. *Therefore, assuming* $\alpha = 0.3$, *we conclude that* $sim_1^{TI}(a, b) = 0.549$ *and* $sim_2^{TI}(a, b) = 0.625$.

$$sim_3^{TI}(a, b) = \frac{|[\![a]\!] \cap [\![b]\!]|}{|[\![a]\!] \cup [\![b]\!]|} \tag{5.22}$$

Intuitively, sim_3^{TI} is the ratio of the number of time units in the intersection of the denotations of a and b to the number of time units in the union.

$$\begin{aligned} sim_4^{TI}(a, b) &= \frac{|[\![a]\!] \cap [\![b]\!]|}{\max_{t \in \{a,b\}} |[\![t]\!]|} & (5.23) \\ sim_5^{TI}(a, b) &= e^{-\alpha \cdot H([\![a]\!], [\![b]\!])} & (5.24) \end{aligned}$$

where $H(P, Q) = \max(h(P, Q), h(Q, P))$, with $P, Q \in \wp(\mathbb{Z})$, is the Hausdorff distance.

$$sim_6^{TI}(a, b) = e^{-\alpha \cdot |c([\![a]\!]) - c([\![b]\!])|} \cdot e^{-\beta \cdot (1 - o([\![a]\!], [\![b]\!]))} \tag{5.25}$$

where $c()$ and α have the same meaning as in Equation 5.20, $o([\![a]\!], [\![b]\!]) = \frac{|[\![a]\!] \cap [\![b]\!]|}{|[\![a]\!] \cup [\![b]\!]|}$ is the amount of overlap between $[\![a]\!]$ and $[\![b]\!]$, and β is a factor that controls how fast the similarity decreases as the amount of overlap between the two time intervals decreases.

Example 49. *Consider again the two terms of sort* $Time\ Interval$ *of Example 48. We observe that* $|[\![a]\!]| = 9$, $|[\![b]\!]| = 3$, $|[\![a]\!] \cap [\![b]\!]| = 3$, *and* $|[\![a]\!] \cup [\![b]\!]| = 9$. *Therefore,*

[11] Since we are assuming a time granularity of days, we are abusing notation and using 05/13/09 instead of the corresponding value $z \in \mathbb{Z}$.

$sim_3^{TI}(a,b) = 0.333$ *and* $sim_4^{TI}(a,b) = 0.333$. *In addition,* $H([\![a]\!],[\![b]\!]) = 5$, *which implies* $sim_5^{TI}(a,b) = 0.22$ *and* $sim_6^{TI}(a,b) = 0.469$, *when* $\alpha = 0.045$ *and* $\beta = 1$.

Sort $NumericInterval$

Consider the sort $NumericInterval$, with denotation universe $\mathcal{U}_{NumericInterval} = \{I \in \wp(\mathbb{N}) \mid I \text{ is connected}\}$[12]. As in the case of the sort $Time\ Interval$, given two terms a and b of sort $NumericInterval$, the similarity between a and b can be defined in any of the following ways.

$$sim_1^{NI}(a,b) = e^{-\alpha \cdot |c([\![a]\!]) - c([\![b]\!])|} \tag{5.26}$$

where, for each numeric interval $t \in dom(NumericInterval)$, $c([\![t]\!]) = \text{avg}_{n \in [\![t]\!]}\, n$ is the center of $[\![t]\!]$, and α is a factor that controls how fast the similarity decreases as the distance between the centers of the two numeric intervals increases.

$$sim_2^{NI}(a,b) = \frac{1}{1 + \alpha \cdot |c([\![a]\!]) - c([\![b]\!])|} \tag{5.27}$$

where $c()$ and α have the same meaning as in Equation 5.26

Example 50. *Consider the two terms of sort NumericInterval* $a =$ *"between 10 and 20" and* $b =$ *"at least 16", and assume that the denotation of* $between$ *and* at_least *are those shown in Table 5.2, with* $\epsilon = 0.1$ *and* $\epsilon = 0.5$ *respectively. Then,* $[\![a]\!]$ *is the interval* $[9, 22]$ *and* $[\![b]\!]$ *is the interval* $[16, 24]$. *We have* $c([\![a]\!]) = 16$ *and* $c([\![b]\!]) = 20$. *Therefore, for* $\alpha = 0.3$, $sim_1^{NI}(a,b) = 0.301$ *and* $sim_2^{NI}(a,b) = 0.455$.

[12] This seems to be a natural denotation for indeterminate expressions such as "between 3 and 6", "more than 3", etc. An exact quantity can be also represented as a singleton.

$$\begin{aligned} sim_3^{NI}(a,b) &= \frac{|[\![a]\!] \cap [\![b]\!]|}{|[\![a]\!] \cup [\![b]\!]|} && (5.28) \\ sim_4^{NI}(a,b) &= \frac{|[\![a]\!] \cap [\![b]\!]|}{\max_{t \in \{a,b\}} |[\![t]\!]|} && (5.29) \\ sim_5^{NI}(a,b) &= e^{-\alpha \cdot H([\![a]\!],[\![b]\!])} && (5.30) \end{aligned}$$

where $H(P, Q)$ is the Hausdorff distance.

$$sim_6^{NI}(a,b) = e^{-\alpha \cdot |c([\![a]\!]) - c([\![b]\!])|} \cdot e^{-\beta \cdot (1 - o([\![a]\!],[\![b]\!]))} \qquad (5.31)$$

where $c()$ and α have the same meaning as in Equation 5.26, $o([\![a]\!], [\![b]\!]) = \frac{|[\![a]\!] \cap [\![b]\!]|}{|[\![a]\!] \cup [\![b]\!]|}$ is the amount of overlap between $[\![a]\!]$ and $[\![b]\!]$, and β controls how fast the similarity decreases as the amount of overlap between the two numeric intervals decreases.

Example 51. *Consider again the two terms of sort $NumericInterval$ of Example 50. We observe that $|[\![a]\!]| = 14$, $|[\![b]\!]| = 9$, $|[\![a]\!] \cap [\![b]\!]| = 7$, and $|[\![a]\!] \cup [\![b]\!]| = 16$. Therefore, $sim_3^{NI}(a,b) = 0.438$ and $sim_4^{NI}(a,b) = 0.5$. Moreover, $H([\![a]\!], [\![b]\!]) = 7$, which implies $sim_5^{NI}(a,b) = 0.122$ and $sim_6^{NI}(a,b) = 0.447$, when $\alpha = 0.045$ and $\beta = 1$.*

5.5 PLINI Probabilistic Logic Programs

In this section, we define the concept of a PLINI-rule and a PLINI-program. Informally speaking, a PLINI-rule states that when certain similarity-based conditions associated with two events e_1, e_2 are true, then the two events are equivalent with some probability. Thus, PLINI-rules can be used to determine when two event descriptions refer to

Event name	***Property***	***Value***
Event1	*date*	02/28/2005
	location	Hillah
	number_of_victims	125
	weapon	car bomb
Event2	*location*	Hilla , south of Baghdad
	number_of_victims	*at_Least* 114
	victim	people
	weapon	massive car bomb
Event3	*killer*	twin suicide attack
	location	town of Hilla
	number_of_victims	*at_least* 90
	victim	Shia pilgrims
Event4	*date*	02/28/2005
	weapon	suicide car bomb
	location	Hilla
	number_of_victims	125
Event5	*killer*	suicide car bomber
	location	Hillah
	number_of_victims	*at_least* 100
Event6	*location*	Hillah
	number_of_victims	125
	victim	Iraqis
	weapon	suicide bomb
Event7	*weapon*	suicide bombs
	location	Hilla south of Baghdad
	number_of_victims	*at_least* 27
Event8	*date*	2005/02/28
	location	Hilla
	number_of_victims	*between* 136 *and* 135
	victim	people queuing to obtain medical identification cards
	weapon	suicide car bomb
Event9	*date*	2005/03/28
	location	Between Hillah and Karbala
	number_of_victims	*between* 6 *and* 7
	victim	Shiite pilgrims
	weapon	Suicide car bomb

Table 5.6: Example of Event Database extracted from news sources

the same event, and when two event descriptions refer to different events. PLINI-rules are variants of annotated logic programs [KS92] augmented with methods to handle similarity between events, as well as similarities between properties of events. Table 5.6 shows a small event database that was automatically extracted from news data by the T-REX system [AS07]. We see here that an event can be represented as a set of (property,value) pairs. Throughout this chapter, we assume the existence of some set $\mathcal{E}$ of event names.

Definition 45. *An* event pair *over sort s is a pair (p, v) where p is a property of sort s and $v \in dom(s)$. An* event *is a pair (e, EP) where $e \in \mathcal{E}$ is an event name and EP is a finite set of event pairs such that each event pair $ep \in EP$ is over some sort $s \in \mathcal{S}$.*

We assume that a set $\mathcal{A}$ of properties is given and use the notation $eventname.property$ to refer to the property of an event. We start by defining event-terms.

Definition 46 (Event-Term)**.** *Suppose $\mathcal{E}$ is a finite set of event names and $\mathcal{V}$ is a possibly infinite set of variable symbols. An* event-term *is any member of $\mathcal{E} \cup \mathcal{V}$.*

Example 52. *Consider the event e_{S3} presented in Section 5.2. Both e_{S3} and v, where v is a variable symbol, are event-terms.*

We now define the concept of an equivalence atom. Intuitively, an equivalence atom says that two events (or properties of events) are equivalent.

Definition 47 (Equivalence Atom)**.** *An* equivalence atom *is an expression of the form*

- *$e_i \equiv e_j$, where e_i and e_j are event-terms, or*
- *$e_i.a_k \equiv e_j.a_l$, where e_i, e_j are event-terms, $a_k, a_l \in \mathcal{A}$, and a_k, a_l are both of sort $s \in \mathcal{S}$, or*
- *$e_i.a_k \equiv b$, where e_i is an event-term, $a_k \in \mathcal{A}$ is an attribute whose associated sort is $s \in \mathcal{S}$ and b a ground term of sort s.*

Example 53. *Let us return to the case of the events e_{S1}, e_{S2}, e_{S3} from Table 5.1. Some example equivalence atoms include:*

$$\begin{aligned} e_{S1} &\equiv e_{S2}. \\ e_{S1}.place &\equiv e_{S3}.place \\ e_{S3}.place &\equiv Ahmedabad. \end{aligned}$$

Note that two events need not be exactly identical in order for them to be considered equivalent. For instance, consider the events e_{S1}, e_{S2}, e_{S3} given in Section 5.2. It is clear that we want these three events to be considered equivalent, even though their associated event pairs are somewhat different. In order to achieve this, we first need to state what it means for terms over various sorts to be equivalent. This is done via the notion of a PLINI-atom.

Definition 48 (PLINI-atom)**.** *If A is an equivalence atom and $\mu \in [0,1]$, then $A : \mu$ is a PLINI-atom.*

The intuitive meaning of a PLINI-atom can be best illustrated via an example.

Example 54. *The PLINI-atom $(e_1.weapon \equiv e_2.weapon) : 0.683$ says that the weapons associated with events e_1 and e_2 are similar with a degree of at least 0.683. Likewise, the PLINI-atom $(e_1.date \equiv e_2.date) : 0.575$ says that the dates associated with events e_1 and e_2 are similar with a degree of at least 0.575.*

When providing a semantics for PLINI, we will use the notion of similarity function for sorts as defined in Section 5.4. There we gave specific examples of similarity functions for the numeric, spatial, and temporal domains. Our theory will be defined in terms of any arbitrary but fixed set of such similarity functions. The heart of our method for identifying inconsistency in news reports is the notion of PLINI-rules which intuitively specify when certain equivalence atoms are true.

Definition 49 (PLINI-rule)**.** *Suppose A is an equivalence atom, $A_1 : \mu_1, \ldots, A_n : \mu_n$ are PLINI-atoms, and $p \in [0,1]$. Then*

$$A \quad \xleftarrow{p} \quad A_1 : \mu_1 \ \wedge \ldots \wedge \ A_n : \mu_n$$

is a PLINI-rule. *If* $n = 0$ *then the rule is called a* PLINI-fact. A *is called the* head *of the rule, while* $A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$ *is called the* body. *A PLINI-rule is* ground *iff it contains no variables.*

Definition 50 (PLINI-program)**.** *A* PLINI-program *is a finite set of PLINI-rules where no rule may appear more than once with different probabilities.*

Note that a PLINI-program is somewhat different in syntax than a probabilistic logic program [NS92] as no probability intervals are involved. However, it is a variant of a generalized annotated program due to [KS92]. In classical logic programming [Llo87], there is a general assumption that logic programs are written by human (logic) programmers. However, in the case of PLINI-programs, they can also be *inferred* automatically from training data. For instance, we learned rules (semi-automatically) to recognize when certain violent events were equivalent to other violent events in the event database generated by the information extraction program T-REX [AS07] mentioned earlier. To do this, we first collected a set of 110 events ("annotation corpus") extracted by T-REX from news events and then manually classified which of the resulting pairs of events from the annotation corpus were equivalent. We then used two classical machine learning programs called JRIP and J48 from the well known WEKA library[13] to learn PLINI-rules automatically from the data. Figure 5.5 shows some of the rules we learned automatically using JRIP.

We briefly explain the first two rules shown in Figure 5.5 that JRIP extracted automatically from the T-REX annotated corpus. The first rule says that when the similarity between the date field of events e_1, e_2 is at least 95.5997%, and when the similarity between the number of victims field of e_1, e_2 is 100%, and the similarity between their location fields is also 100%, then the probability that e_1 and e_2 are equivalent is 100%.

[13] http://www.cs.waikato.ac.nz/ml/weka/

$$\begin{array}{lcl}
e_1 \equiv e_2 & \xleftarrow{1.0} & e_1.date \equiv e_2.date : 0.955997 \ \wedge \\
 & & e_1.number\_of\_victims \equiv e_2.number\_of\_victims : 1 \ \wedge \\
 & & e_1.location \equiv e_2.location : 1. \\
e_1 \equiv e_2 & \xleftarrow{0.75} & e_1.date \equiv e_2.date : 1 \ \wedge \ e_1.killer \equiv e_2.killer : 0.574707. \\
e_1 \equiv e_2 & \xleftarrow{0.5833} & e_1.date \equiv e_2.date : 1 \ \wedge \\
 & & e_1.weapon \equiv e_2.weapon : 0.634663 \ \wedge \\
 & & e_1.location \equiv e_2.location : 1.
\end{array}$$

Figure 5.5: Some automatically learned PLINI-rules from T-REX data using JRIP

The second rule says that when the dates of events e_1, e_2 are 100% similar, and the killer fields are at least 57.4707% similar, then the events are at least 75% similar.

We see from this example that PLINI-programs weave together notions of similarity from different domains (within the annotations of equivalence atoms in the rule body) and the notion of probability attached to a rule. We now recall the standard concept of a substitution.

Definition 51 (Substitution). *Suppose R is a PLINI-rule. A substitution $\sigma = [X_1/e_1, \ldots, X_n/e_n]$ for R is a finite set of pairs of terms where each e_i is an event-term and $X_i \neq X_j$ when $i \neq j$.*

A ground instance of R under σ is the result of simultaneously replacing all variables X_i in R with the event-term e_i where $X_i/e_i \in \sigma$.

5.6 Model Theory and Fixpoint Theory

In this section, we specify a formal model theory for PLINI-programs by leveraging the semantics of generalized annotated programs [KS92]. For each sort $s \in \mathcal{S}$, we assume the existence of a similarity function $sim_s : dom(s) \times dom(s) \to [0, 1]$. Intuitively,

$sim_s(v_1, v_2)$ returns 0 if domain values v_1, v_2 are completely different and returns 1 if the two values are considered to be the same. We have already provided many possible definitions for similarity functions in Section 5.4. We first need to define the Herbrand Base.

Definition 52 (Herbrand Base)**.** *$\mathcal{B}_\mathcal{E}$ is the set of* all ground equivalence atoms *that can be formed from the event-terms, attributes, and constant symbols associated with $\mathcal{E}$.*

Clearly, $\mathcal{B}_\mathcal{E}$ is finite. We now define the concept of an interpretation.

Definition 53 (Interpretation)**.** *Any function $I : \mathcal{B}_\mathcal{E} \rightarrow [0, 1]$ is called an* interpretation.

Thus, an interpretation just assigns a number in $[0, 1]$ to each ground equivalence atom. We now define satisfaction of PLINI-rules by interpretations.

Definition 54 (Satisfaction)**.** *Let I be a interpretation, and let $A, A_1, \ldots, A_n \in \mathcal{B}_\mathcal{E}$. Then:*

- *$I \models A : \mu$ iff $I(A) \geq \mu$.*
- *$I \models A_1 : \mu_1 \ \wedge \ldots \wedge \ A_n : \mu_n$ iff $I \models A_i : \mu_i$ for all $1 \leq i \leq n$.*
- *$I \models A \xleftarrow{p} A_1 : \mu_1 \ \wedge \ldots \wedge \ A_n : \mu_n$ iff either $I \not\models A_1 : \mu_1 \ \wedge \ldots \wedge \ A_n : \mu_n$ or $I(A) \geq p\}$.*

I satisfies a non-ground rule iff it satisfies all ground instances of the rule. I satisfies a PLINI-program Π iff it satisfies all PLINI-rules in Π.

The first part of the above definition says that for $A : \mu$ to be true w.r.t. an interpretation I, we should just check that $I(A)$ is greater than or equal to μ. Satisfaction of conjunctions is defined in the obvious way. Satisfaction of a ground PLINI-rule is defined in a more complex way. Either the body of the rule should be false with respect to I or I must assign a value at least p to the head. As usual, $A : \mu$ is a *logical consequence* of Π iff every interpretation that satisfies Π also satisfies $A : \mu$.

Definition 55. *Suppose Π is a PLINI-program and we have a fixed set of similarity functions sim_s for each sort s. The* augmentation of Π with similarity information *is the PLINI-program $\Pi^{sim} = \Pi \cup \{(x \equiv y) \overset{sim_s(x,y)}{\longleftarrow} \mid x, y$ are ground terms of sort s in $\Pi\}$.*

Throughout the rest of this chapter, we only consider the augmented program Π^{sim}. We are interested in characterizing the set of ground equivalence atoms that are logical consequence of Π^{sim}. Given a PLINI-program Π, we are now ready to associate with Π, a fixpoint operator T_Π which maps interpretations to interpretations.

Definition 56. $T_\Pi(I)(A) = A : \mathbf{sup}\{p|\ A \overset{p}{\longleftarrow} A_1 : \mu_1\ \wedge \ldots \wedge\ A_n : \mu_n$ *is a ground instance of a rule in Π and for all $1 \leq i \leq n$, either $I(A_i) \geq \mu_i$ or A_i has the form $x_i \equiv y_i$ and $sim_s(x_i, y_i) \geq \mu_i\}$.*

The above definition says that in order to find the truth value assigned to a ground atom A by the interpretation $T_\Pi(I)$, we first need to look at all rules in Π that have A as the head of a ground instance of that rule. To check whether the body of such a rule is true w.r.t. I, we need to look at each ground equivalence atom $A_i : \mu_i$ where A_i has the form $(x_i \equiv y_i)$. This atom is satisfied if either $I(x_i \equiv y_i)$ is greater than or equal to μ_i or if the similarity function for sort s (of the type of x_i, y_i) assigns a value greater than or equal to μ_i to (x_i, y_i). Note that the T_Π operator operates on the Π^{sim} program without explicitly adding equivalence atoms of the form $(x \equiv y) \overset{sim_s(x,y)}{\longleftarrow}$ to Π, thus ensuring a potentially large saving.

It is easy to see that the set of all interpretations forms a complete lattice under the following ordering: $I_1 \leq I_2$ iff for all ground equivalence atoms A, $I_1(A) \leq I_2(A)$. We can define the *powers* of T_Π as follows.

$$\begin{aligned} T_\Pi \uparrow 0(A) &= A : 0 \text{ for all ground equivalence atoms A.} \\ T_\Pi \uparrow (j+1)(A) &= (T_\Pi(T_\Pi \uparrow j))(A). \\ T_\Pi \uparrow \omega(A) &= \bigcap \{T_\Pi \uparrow j(A) \mid j \geq 0\}. \end{aligned}$$

The result below follows directly from similar results for generalized annotated programs [KS92] and shows that the T_Π operator has some nice properties.

Proposition 26. *Suppose* Π *is a PLINI-program and* sim_s *is a family of similarity functions for a given set of sorts. Then:*

1. T_Π *is monotonic, i.e.* $I_1 \leq I_2 \rightarrow T_\Pi(I_1) \leq T_\Pi(I_2)$.

2. T_Π *has a least fixpoint, denoted* $lfp(T_\Pi)$ *which coincides with* $T_\Pi \uparrow \omega$.

3. I *satisfies* Π^{sim} *iff* $T_\Pi(I) = I$.

4. $A : \mu$ *is a logical consequence of* Π^{sim} *iff* $lfp(T_\Pi)(A) \geq \mu$.

5.7 Event Clustering Algorithm

Suppose e_1, e_2 are any two reported events. The least fixpoint of the T_Π operator gives the probability that the two events are equivalent (i.e., they refer to the same real-world event). Alternatively, $lfp(T_\Pi)(e_1 \equiv e_2)$ can be interpreted as the similarity between e_1 and e_2, meaning that if the similarity between two events is high, they are likely to refer to the same real-world event. In other words, the least fixpoint of the T_Π operator gives us some information on the pairwise similarity between events. However, we may have a situation where the similarity according to $lfp(T_\Pi)$ between events e_1 and e_2 is 0.9, between e_2, and e_3 is 0.7, but the similarity between e_1 and e_3 is 0.5. In general, given a finite set $\mathcal{E}$ of events, we would like to look at the results computed by $lfp(T_\Pi)$, and cluster the events into *buckets* of equivalent events. We then need to find a partition $\mathcal{P} = \{P_1, \ldots, P_k\}$ of $\mathcal{E}$, such that similar events are assigned to the same partition and dissimilar events are assigned to different partitions. In this section, we define the PLINI-Cluster algorithm, which can find a sub-optimal solution – w.r.t. the score function defined below – in polynomial time.

Definition 57 (Score of Event Partition)**.** *Let* Π *be a* PLINI*-program and* $\tau \in [0,1]$ *a threshold. Let* $\mathcal{E}$ *be a set of events and* $\mathcal{P} = \{P_1, \ldots, P_k\}$ *a partition of* $\mathcal{E}$*, i.e.* $\bigcup_{i=1}^{k} P_i = \mathcal{E}$*, and* $(\forall i \neq j)\ P_i \cap P_j = \emptyset$*. We define the score of partition* $\mathcal{P}$ *as*

$$S(\mathcal{P}) = S_i(\mathcal{P}) + S_e(\mathcal{P})$$

where $S_i(\mathcal{P})$ *is the* internal score *of partition* $\mathcal{P}$ *given by*

$$S_i(\mathcal{P}) = \sum_{P_j \in \mathcal{P}} \sum_{e_r, e_s \in P_j, e_r \neq e_s} (lfp(T_\Pi)(e_r \equiv e_s) - \tau)$$

and $S_e(\mathcal{P})$ *is the* external score *of partition* $\mathcal{P}$ *given by*

$$S_e(\mathcal{P}) = \sum_{e_r, e_s \in \mathcal{E}, e_r \in P_u, e_s \in P_v, u \neq v} (\tau - lfp(T_\Pi)(e_r \equiv e_s))$$

Intuitively, $S_i(\mathcal{P})$ measures the similarity between objects within a partition component. Two events e_1, e_2 in the same cluster contribute positively to the internal score if their similarity is above the threshold τ. Instead, if their similarity is below the threshold the partition is penalized. Analogously, $S_e(\mathcal{P})$ measures the similarity across multiple partition components. The external score of a partition is higher when events in different clusters have lower similarity. Two events e_1, e_2 in different clusters contribute positively to the external score if their similarity is below the threshold τ.

Finding the partition $\mathcal{P} = \{P_1, \ldots, P_k\}$ which maximizes the score S defined above gives rise to a combinatorial optimization problem. Note that k is a problem parameter and not known a priori, in contrast to common clustering problems in machine learning and other areas. Ozcan et al. [OS04] were the first to determine how to partition a set of entities under this intuition – they did so in the context of partitioning a set of activities an agent is supposed to perform so that the activities in each component of the partition could be executed by leveraging commonalities of tasks amongst those activities. They proved that the partitioning problem was NP-hard and provided efficient heuristics to solve the problem. Later, Bansal et al. [BBC04] showed that finding the optimal partition with respect to the score S is NP-complete. Demaine and Immorlica [DI03] presented an efficiently computable $O(\log n)$ approximation based on integer program relaxation and proved that the bound is optimal for such relaxations. We now present a simple, greedy, hill-climbing algorithm for this task (Algorithm 5.6). The algorithm starts by assigning each event to a different cluster. Then, at each iteration, it merges the two clusters that provide the highest increase of the score. It stops when no further increase of the score can be achieved by merging two clusters.

Example 55. *Suppose we have 4 events e_1, e_2, e_3, e_4, and assume $\tau = 0.5$. Table 5.7 shows the degree of similarity for each pair of events according to some fixed PLINI-program Π, i.e. the values of $lfp(T_\Pi)(e_i \equiv e_j)$. The algorithm starts by initializing $P_1 = \{e_1\}$, $P_2 = \{e_2\}$, $P_3 = \{e_3\}$, $P_4 = \{e_4\}$. In the first iteration, P_1 and P_3 are*

```
Algorithm PLINI-Cluster
Input: PLINI-program $\Pi$; Set of Events $\mathcal{E}$; Threshold $\tau$

$\mathcal{P} \leftarrow \emptyset$
For all $e_i \in \mathcal{E}$,
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\{e_i\}\}$
Repeat
  For all $P_i, P_j \in \mathcal{P}$ s.t. $i \neq j$,
    /*Compute the change in the score of the partition, if we merge clusters $P_i$, $P_j$*/
    $s_{i,j} \leftarrow S\left((\mathcal{P} \setminus \{P_i, P_j\}) \cup \{P_i \cup P_j\}\right) - S(\mathcal{P}) =$
      $= 2 \cdot \sum_{e_r \in P_i, e_s \in P_j} (lfp(T_\Pi)(e_r \equiv e_s) - \tau)$ /*Note that $s_{i,j} = s_{j,i}$*/
  $s_{u,v} \leftarrow \max_{i,j \in [1,|\mathcal{P}|] \wedge i \neq j} \{s_{i,j}\}$
  If ($s_{u,v} > 0$) then
    $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{P_u, P_v\}) \cup \{P_u \cup P_v\}$
Until ($s_{u,v} \leq 0$)
```

Figure 5.6: Algorithm PLINI-Cluster$(\Pi, \mathcal{E}, \tau)$

	e_1	e_2	e_3	e_4
e_1	1	0.2	0.9	0.3
e_2		1	0	0.8
e_3			1	0.6
e_4				1

Table 5.7: Degrees of similarity as given by $lfp(T_\Pi)(e_i \equiv e_j)$

merged, because this leads to the largest increase in $S(\mathcal{P})$, $s_{1,3} = 0.8$. *In the second iteration,* P_2 *and* P_4 *are merged for the same reason, with* $s_{2,4} = 0.6$. *In the third iteration, the algorithm terminates because no score increase can be achieved, as merging the remaining two clusters would decrease the score by* 1.8. *Hence, the resulting partition is* $\{\{e_1, e_3\}, \{e_2, e_4\}\}$.

The following result shows that the above algorithm finds the locally optimal partition w.r.t the cost function and that its worst case runtime complexity is $O(n^3)$.

Proposition 27. *Suppose* Π *is a PLINI-program,* $\mathcal{E}$ *a set of events, and* τ *a threshold. Then PLINI-Cluster*$(\Pi, \mathcal{E}, \tau)$ *finds a locally optimal partition of* $\mathcal{E}$ *w.r.t. to the score function* S*, and its worst case complexity is* $O(n^3)$ *where* n *is the total number of events.*

Proof. Let n denote the total number of events. We assume that we have an oracle to look up or compute $lfp(T_\Pi)$ for any pair of events in constant time. During each iteration of the algorithm either two clusters are merged or the algorithm terminates. The algorithm starts with a partition of size n and since a merger reduces the partition size by one, there can be at most n iterations. Now we turn to the complexity of each iteration, namely the cost of computing the $s_{i,j}$'s. For any given iteration, we have at most $O(n)$ clusters of constant size and a constant number of clusters of size $O(n)$, since the total number of elements (i.e. events) is n. If $|P_i| \in O(1)$ and $|P_j| \in O(1)$, then the cost of computing $s_{i,j}$ is $O(1)$ as well. Similarly, if $|P_i| \in O(n)$ and $|P_j| \in O(1)$, this cost is $O(n)$ and if $|P_i| \in O(n)$ and $|P_j| \in O(n)$ the cost is $O(n^2)$. Since we compute $s_{i.j}$ for all pairs of clusters, the overall complexity is

$$O(n) \times O(n) \times O(1) + O(n) \times O(1) \times O(n) + O(1) \times O(1) \times O(n^2) = O(n^2)$$

where the first summand is the complexity for pairs of constant size partitions, the second summand for pairs of linear with constant size partitions, and the last summand for pairs of linear size partitions. Hence, the complexity of each iteration is $O(n^2)$ and therefore the overall runtime complexity of the event clustering algorithm is in $O(n^3)$. □

Note that due to the sparsity of event similarities in real world datasets, we can effectively prune a large number of partition comparisons. We can prune the search space for the optimal merger even further, by considering highly associated partitions first. These optimizations do not impact the worst case runtime complexity, but render our algorithm very efficient in practice.

5.8 Implementation and Experiments

Our experimental prototype PLINI system was implemented in approximately 5700 lines of Java code. In order to test the accuracy of PLINI, we developed a training data set and a separate evaluation data set. We randomly selected a set of 110 event descriptions from the millions automatically extracted from news sources by T-REX [AS07]. We then generated all the 5,995 possible pairs of events from this set and asked human reviewers to judge the equivalence of each such pair. The ground truth provided by the reviewers was used to learn PLINI-programs for different combinations of learning algorithms and similarity functions. Specifically, we considered 588 different combinations of similarity functions and learned the corresponding 588 PLINI-programs using both JRIP and J48. The evaluation data set was similarly created by selecting 240 event descriptions from those extracted by T-REX.

All experiments were run on a machine with multiple, multi-core Intel Xeon E5345 processors at 2.33GHz, 8GB of memory, running the Scientific Linux distribution of the GNU/Linux operating system. However, the current implementation has not been parallelized and uses only one processor and one core at a time.

PLINI-programs corresponding to each combination of algorithms and similarity functions were run on the entire set of 28,680 possible pairs of events in the test set. However, evaluation was conducted on subsets of pairs of a manageable size for human reviewers. Specifically, we selected 3 human evaluators and assigned each of them two subsets of pairs to evaluate. The first subset was common to all 3 reviewers and included 50 pairs that at least one program judged equivalent with confidence greater than 0.6 (i.e. T_{Π} returned over 0.6 for these pairs) and 100 pairs that no program judged equivalent with probability greater than 0.6. The second subset was different for each reviewer, and included 150 pairs, selected in the same way as the first set. Thus, altogether we evaluated a total of 600 distinct pairs.

We then computed precision and recall as defined below. Suppose $\mathcal{E}_p$ is the set of event pairs being evaluated. We use $e_1 \equiv_h e_2$, to denote that events e_1 and e_2 were judged to be equivalent by a human reviewer. We use $P(e_1 \equiv e_2)$ to denote the probability assigned by the algorithm to the equivalence atom $e_1 \equiv e_2$. Given a threshold value $\tau \in [0, 1]$, we define the following sets.

- $\mathcal{TP}_1^\tau = \{(e_1, e_2) \in \mathcal{E}_p | P(e_1 \equiv e_2) \geq \tau \wedge e_1 \equiv_h e_2\}$ is the set of pairs flagged as equivalent (probability greater than the threshold τ) by the algorithm and actually judged equivalent by human reviewers;

- $\mathcal{TP}_0^\tau = \{(e_1, e_2) \in \mathcal{E}_p | P(e_1 \equiv e_2) < \tau \wedge e_1 \not\equiv_h e_2\}$ is the set of pairs flagged as not equivalent by the algorithm and actually judged not equivalent by human reviewers;

- $\mathcal{P}_1^\tau = \{(e_1, e_2) \in \mathcal{E}_p | P(e_1 \equiv e_2) \geq \tau\}$ is the set of pairs flagged as equivalent by the algorithm;

- $\mathcal{P}_0^\tau = \{(e_1, e_2) \in \mathcal{E}_p | P(e_1 \equiv e_2) < \tau\}$ is the set of pairs flagged as not equivalent by the algorithm;

Given a threshold value $\tau \in [0, 1]$, we define precision, recall, and F-measure as follows.

$$P_1^\tau = \frac{|\mathcal{TP}_1^\tau|}{|\mathcal{P}_1^\tau|} \qquad P_0^\tau = \frac{|\mathcal{TP}_0^\tau|}{|\mathcal{P}_0^\tau|} \qquad P^\tau = \frac{|\mathcal{TP}_1^\tau| + |\mathcal{TP}_0^\tau|}{|\mathcal{E}_p|}$$

$$R_1^\tau = \frac{|\mathcal{TP}_1^\tau|}{|\{(e_1, e_2) \in \mathcal{E}_p | e_1 \equiv_h e_2\}|}^{14} \qquad F^\tau = \frac{2 \cdot P_1^\tau \cdot R_1^\tau}{P_1^\tau + R_1^\tau}$$

[14] Given the nature of the problem, most pairs of event descriptions are not equivalent. Therefore, the best indicators of our system performance are recall/precision w.r.t. equivalent pairs.

τ	P_1^τ	P_0^τ	P^τ	R_1^τ	F^τ
0.50	83.5%	88.6%	88.2%	36.9%	0.512
0.60	84.4%	88.5%	88.2%	36.5%	0.510
0.70	84.5%	88.5%	88.2%	36.4%	0.509
0.80	89.7%	87.9%	88.1%	32.5%	0.477
0.90	92.3%	87.3%	87.6%	28.2%	0.432
0.95	91.8%	86.5%	86.7%	22.7%	0.364

Table 5.8: Performance of J48 for different values of τ

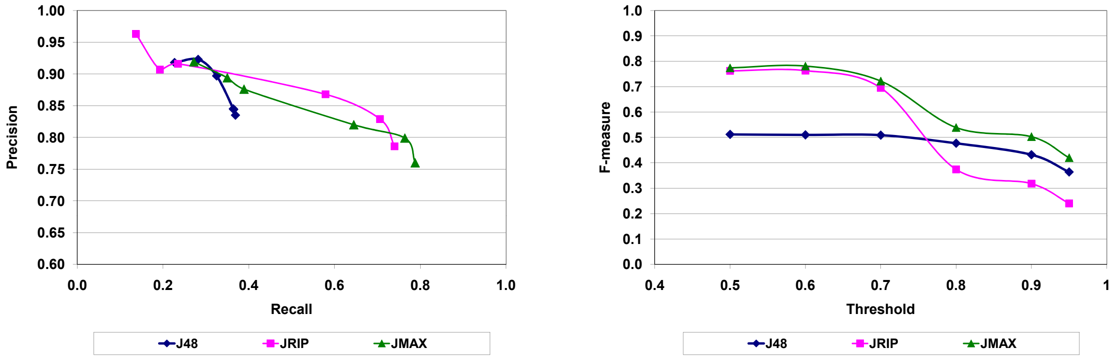
τ	P_1^τ	P_0^τ	P^τ	R_1^τ	F^τ
0.50	78.6%	94.8%	92.3%	74.0%	0.762
0.60	82.9%	94.2%	92.6%	70.6%	0.763
0.70	86.8%	92.0%	91.4%	57.9%	0.695
0.80	91.6%	86.6%	86.8%	23.5%	0.374
0.90	90.7%	86.0%	86.1%	19.3%	0.318
0.95	96.3%	85.2%	85.4%	13.7%	0.240

Table 5.9: Performance of JRIP for different values of τ

Note that all the quality measures defined above are parameterized by the threshold τ.

Accuracy results. Tables 5.8 and 5.9 report the overall performance of the PLINI-programs derived using J48 and JRIP, respectively for different values of the threshold τ. Performance measures are averaged over all the 588 combinations of similarity metrics and over all the reviewers. Figure 5.7(a) shows precision/recall curves for both algorithms, while Figure 5.7(b) plots the F-measure vs. different values of τ.

As expected, when τ increases P_1^τ increases and R_1^τ decreases. However, R_1^τ decreases more rapidly than the increase in P_1^τ, causing F^τ to degrade. In general, rules extracted with JRIP outperform those extracted using J48, but it is also clear that the performance of JRIP in terms of F-measure degrades more rapidly than J48 due to a drastic drop in recall for higher thresholds. In fact, for thresholds above 0.75, J48-derived PLINI-programs surpass JRIP-derived PLINI-programs in terms of F-measure. Also note



(a)

(b)

Figure 5.7: (a) Recall/Precision and (b) F-measure for J48 and JRIP

that when τ increases P_0^τ decreases. This is because higher thresholds cause a larger number of equivalent pairs to be classified as non equivalent, therefore the fraction of pairs flagged as not equivalent which are actually not equivalent decreases. However, precision is very high. When J48 and JRIP are used, the optimal value of the F-measure is achieved for $\tau = 0.6$, which corresponds to 84% precision and 37% recall for J48, and 83% precision and 71% recall for JRIP.

JMAX (which we defined) assigns to each equivalence atom the maximum of the probabilities assigned by JRIP-derived rules and J48-derived PLINI-rules. Figure 5.7(a) shows that JMAX produces significantly higher recall at the expense of a slight decrease in precision. The overall gain in performance is more evident in Figure 5.7(b), which shows that the F-measure for JMAX is higher than that for both J48 or JRIP for virtually all values of the threshold τ. The optimal value of F is achieved for $\tau = 0.6$, which corresponds to 80% precision and 75% recall.

Table 5.10 reports the average performance of PLINI-programs derived using the 3 algorithms – J48, JRIP, JMAX when individually compared with the ground truth provided by each of the 3 reviewers enrolled for the evaluation. There is no significant

Algorithm	Reviewer	P_1^τ	P_0^τ	P^τ	R_1^τ	F^τ
J48	1 2 3	87.6% 74.4% 90.7%	87.9% 89.9% 87.7%	87.9% 88.8% 87.9%	33.6% 36.1% 39.3%	0.486 0.486 0.548
JRIP	1 2 3	84.6% 80.2% 83.9%	93.9% 95.9% 92.9%	92.6% 93.7% 91.6%	68.8% 76.1% 67.8%	0.759 0.781 0.750
JMAX	1 2 3	82.5% 74.3% 82.9%	94.8% 96.5% 94.6%	92.9% 93.0% 92.6%	73.8% 80.2% 75.8%	0.779 0.771 0.792

Table 5.10: Average performance of JRIP for $\tau = 0.6$ when compared with different revilers

difference between the reviewers and, in fact, they unanimously agreed in 138 out of the 150 common cases (92%).

We found that in general using both J48-based PLINI-rules and JRIP-based PLINI-rules (encompassed in our JMAX strategy) offers the best performance, while using only J48-derived PLINI-rules is the worst.

5.9 Concluding Remarks

The number of "formal" news sources on the Internet is mushrooming rapidly. Google News alone covers thousands of news sources from around the world. If one adds consumer generated content and informal news channels run by individual amateur newsmen and women who publish blogs about local or global items of interest, the number of news sources reaches staggering numbers. As shown in the Introduction, inconsistencies can occur for many reasons.

The goal of this work is not to resolve these inconsistencies, but to identify when event data reported in news sources is inconsistent. When information extraction programs are used to automatically mine event data from news information, the resulting properties of the events extracted are often linguistically qualified. In this chapter, we have studied three kinds of linguistic modifiers typically used when such programs are used – linguistic modifiers applied to numbers, spatial information, and temporal information. In each case, we have given a formal semantics to a number of linguistically modified terms.

In order to determine whether two events described in one or more news sources are the same, we need to be able to compare the attributes of these two events. This is done via similarity measures. Though similarity measures for numbers are readily available, no formal similarity mechanisms exist (to the best of our knowledge) for linguistically-modified numbers. The same situation occurs in the case of linguistically-modified temporal information and linguistically modified geospatial information. We provide formal definitions of similarity for many commonly used linguistically modified numeric, temporal, and spatial information.

We subsequently introduce PLINI-programs as a variant of the well known generalized annotated program (GAP) [KS92] framework. PLINI-programs can be learned automatically from a relatively small annotated corpus (as we showed) using standard machine learning algorithms like J48 and JRIP from the WEKA library. Using PLINI-programs, we showed that the least fixpoint of an operator associated with PLINI-programs tells us the degree of similarity between two events. Once such a least fixpoint has been computed, we present the PLINI-Cluster algorithm to cluster together sets of events that are similar, and sets of events that are dissimilar.

We have experimentally evaluated our PLINI-framework using many different similarity functions (for different sorts), many different threshold values, and three alternative ways of automatically deriving PLINI-programs from a small training corpus. Our experiments show that the PLINI-framework produced high precision and recall when compared with human users evaluating whether two reports talked about the same event or not.

There is much work to be done in the future. PLINI-programs do not include negation. A sort of stable models semantics [GL98] can be defined for PLINI-programs that include negation. However, the challenge will be to derive such programs automatically from a training corpus (standard machine learning algorithms do not do this) and to apply them efficiently as we can do with PLINI.

Chapter 6

Partial Information Policies

The work presented in this chapter is taken from [MMGS11].

6.1 Introduction and Motivating Example

Partial information arises when information is unavailable to users of a database when they enter new data. All commercial real-world relational database systems implement some *fixed* way of managing incomplete information; but neither the RDBMS nor the user has any say in how the partial information is interpreted. But does the user of a stock database really expect an RDBMS designer to understand his risks and his mission in managing the incomplete information? Likewise, does an epidemiologist collecting data about some disease have confidence that an RDBMS designer understands how his data was collected, why some data is missing, and what the implications of that missing data are for his disease models and applications? The answer is usually no. While database researchers have understood the diversity of types of missing data [AM86, Bis79, CGS97b, Cod79, Gra91, IL84b, LL98, Zan84] (e.g. a value exists but is *unknown* — this may happen when we known someone has a phone but do not know the number; a value does not exist in a given case because the field in question is *inapplicable* - this may happen when someone does not have a spouse, leading to an inapplicable null in a relation's spouse field; or we have *no-information* about whether a value exists or not - as in the case when we do not know if someone has a cell phone), the SQL standard only supports one type of unmarked null value, so RDBMSs force users to handle all partial information in the same way, even when there are differences.

We have worked with two data sets containing extensive partial information. A data set about education from the World Bank and UNESCO contains data for each of 221 countries with over 4000 attributes per country. As the data was collected manually, there are many incomplete entries. The incompleteness is due to many factors (e.g., conflict in the country which made collecting difficult during certain time frame).

Example 56. *The relation below shows a very small number of attributes associated with Rwanda for which conflict in the 90s led to a lot of incomplete information. The relation only shows a few of the 4000+ attributes (GER and UER stand for gross and under-age enrollment ratio, respectively). Even in this relatively small relation, we see there are a lot of unknown values (here the U_i's denote unknown values).*

Country	*Year*	*% of female unemployment*	*GER*	*UER*	*Net ODA from non-DAC donors (current US$)*
Rwanda	*1995*	U_1	U_{11}	U_{15}	*-260000*
Rwanda	*1996*	*0.0*	U_{12}	U_{16}	*1330000*
Rwanda	*1997*	U_2	U_{13}	U_{17}	*530000*
Rwanda	*1998*	U_3	U_{14}	U_{18}	*130000*
Rwanda	*1999*	U_4	*99.37*	U_{19}	*90000*
Rwanda	*2000*	*9.4*	*103.55*	U_{20}	*170000*
Rwanda	*2001*	U_5	*104.06*	*4.59*	*130000*
Rwanda	*2002*	U_6	*107.77*	*4.76*	*140000*
Rwanda	*2003*	U_7	*116.5*	*4.62*	*110000*
Rwanda	*2004*	U_8	*128.05*	*4.42*	*90000*
Rwanda	*2005*	U_9	*139.21*	*4.81*	*120000*
Rwanda	*2006*	U_{10}	*149.88*	U_{21}	*450000*

Users may want to fill in the missing values in many possible ways. For instance, User A may want to fill the under-age enrollment ratio (UER) column via linear regression. User B may fill in missing values by choosing the interval $[4.81, 16.77]$ *that says the missing value is unknown but lies in this interval. User C may require that the only possible values are the under-age enrollment ratios appearing in the tuples of the relation. User D may want to learn this value by studying its relationship with the ODA from non-DAC donors and extrapolating - this would occur when the user believes the under-age enrollment ratio is correlated with the ODA column and, in this case, he learns that UER is a function of the latter and uses this for extrapolation. User E may want to overestimate a missing UER by replacing it with the maximum UER for the same year from the other countries. User F may want to replace a missing under-age enrollment ratio by looking at the gross enrollment ratios of the other countries for the same year and taking the under-age enrollment ratio corresponding to average gross enrollment ratio. Users may wish to* *specify many other policies based on their application, their mission, their attitude to risk (of being wrong), the expectations of their bosses and customers, and other factors.*

There are many queries of interest that an education analyst may want to pose over the data above. He may be interested in the years during which the % of female unemployment was above a certain threshold and want to know what were the gross and under-age enrollment ratios in those years. He may want to know the countries with the highest average UER in the 90's. It is easy to see that such queries would yield poor results when evaluated on the original database whereas higher quality answers are obtained if the missing values are populated according to the knowledge the user has of the data.

Useful computing systems must support users' desires. Though the database theory literature counts several works on null values (e.g., [AKG91, CGS97b, Gra91, IL84b, Zan84]), all of them provide a fixed "a priori" semantics for nulls, allowing the user none of the flexibility required by users A, B, C, D, E, and F above. Other works in the fields of data mining, data warehousing, and data management, such as [Qui93, MST94, Pyl99, BFOS84, Li09], have proposed *fixed* approaches for replacing nulls that work in specific domains and applications and do not allow modeling different kinds of partial information and different ways of resolving incompleteness. In contrast, we want users to be able to specify policies to manage their partial information and then have the RDBMS directly answer queries in accordance with their PIP.

The principal novelty in this chapter is that partial information policies (PIPs) allow end-users the flexibility to specify how they want to handle partial information, something the above frameworks do not do.

The main contributions of this chapter are the following.

1. We propose the general notion of partial information policy for resolving various kinds of incompleteness and give several useful and intuitive instances of PIPs.

2. We propose index structures to support the efficient application of PIPs and show how to maintain them incrementally as the database is updated.

3. We study the interaction between relational algebra operators and PIPs. Specifically, we identify conditions under which applying PIPs before or after a relational algebra operator yield the same result – this can be exploited for optimization purposes.

4. We experimentally assess the effectiveness of the proposed index structures with a real-world airline data set. Specifically, we compare an algorithm exploiting our index structures with a naive one not relying on them and show that the former greatly outperforms the latter and is able to manage very large databases. Moreover, we experimentally evaluate the effect of the index structures when PIPs are combined with relational algebra operators and study whether applying a policy before or after a relational algebra operator, under the conditions which guarantee the same result, may lead to better performance. Finally, we carry out an experimental assessment of the quality of query answers with and without PIPs.

In classical RDBMS architectures, users specify an SQL query which is typically converted into a relational algebra query. A cost model and a set of query rewrite rules allow an RDBMS query optimizer to rewrite the query into a minimal cost query plan which is then executed. Standard `SELECT` A_1`, ..., `A_k `FROM` R_1`, ..., `R_m `WHERE` $cond_1$`, ..., `$cond_n$ queries can be expanded easily to specify PIPs as well. A possible syntax could be

SELECT A_1, ..., A_k FROM R_1, ..., R_m WHERE $cond_1$, ..., $cond_n$

USING POLICY ρ [LAST|FIRST]

where ρ is one of a library of PIPs in the system. The keyword at the end of the clause will determine the semantics of the policy application. Choosing $FIRST$ yields a *policy first* semantics which would first apply ρ to all relations in the $FROM$ clause and then execute the `SELECT...FROM...WHERE...` part of the query on the modified relation instances. Choosing $LAST$ yields a *policy last* semantics which would first execute the `SELECT...FROM...WHERE...` query and then apply the PIP ρ to the result. We consider both these options in this work.

The rest of the chapter is organized as follows. In Section 6.2, we define the syntax and semantics of databases containing the three types of null values mentioned before. Then, in Section 6.3, we introduce the notion of partial information policy and show different families of PIPs. In Section 6.4, we propose index structures to efficiently apply PIPs. In Section 6.5, we study the interaction between PIPs and relational algebra operators. Section 6.6 reports experimental results.

6.2 Preliminaries

Syntax. We assume the existence of a set $\mathcal{R}$ of *relation symbols* and a set Att of *attribute symbols*. Each relation symbol r has an associated relation *schema* $r(A_1, \dots, A_n)$, where the A_i's are attribute symbols. Each attribute $A \in Att$ has a domain $dom(A)$ containing a distinguished value $\perp$, called *inapplicable null*[1] – in addition, there are two infinite disjoint sets (also disjoint from $dom(A)$) $\mathcal{U}(A)$ and $\mathcal{N}(A)$ of variables associated with A. Intuitively, $\mathcal{U}(A)$ is a set of variables denoting *unknown nulls*, while $\mathcal{N}(A)$ is a set

[1] Note that we treat an inapplicable null as a value in $dom(A)$ since it does not represent uncertain information.

of variables that denote *no-information nulls*. We require that $\mathcal{U}(A) \cap \mathcal{U}(B) = \emptyset$ if $dom(A) \neq dom(B)$ and $\mathcal{U}(A) = \mathcal{U}(B)$ if $dom(A) = dom(B)$, for any $A, B \in Att$. The same assumptions are made for the $\mathcal{N}(A_i)$'s. We define $Dom = \bigcup_{A \in Att} dom(A)$, $\mathcal{U} = \bigcup_{A \in Att} \mathcal{U}(A)$, $\mathcal{N} = \bigcup_{A \in Att} \mathcal{N}(A)$. For each $A \in Att$ we define $dom(A) = dom(A) - \{\bot\}$.

Given a relation schema $S = r(A_1, \ldots, A_n)$, a *tuple* over S is an element of $(dom(A_1) \cup \mathcal{U}(A_1) \cup \mathcal{N}(A_1)) \times \cdots \times (dom(A_n) \cup \mathcal{U}(A_n) \cup \mathcal{N}(A_n))$; a *relation* over S is a finite multiset of tuples over S. A *complete tuple* belongs to $dom(A_1) \times \cdots \times dom(A_n)$ and a relation R is *complete* iff every tuple in R is complete. The *restriction* of a tuple t to a set X of attributes (or a single attribute) is denoted by $t[X]$. The set of attributes of a relation schema S is denoted by $Att(S)$.

A *database schema* DS is a set of relation schemas $\{S_1, \ldots, S_m\}$; a *database instance* (or simply *database*) I over DS is a set of relations $\{R_1, \ldots, R_m\}$, where each R_i is a relation over S_i. The set of all possible databases over a database schema DS is denoted by $db(DS)$. Multiple occurrences of the same null occur in a database.

We consider the relational algebra operators π (projection), σ (selection), $\times$ (cartesian product), $\bowtie$ (join), $\cup$ (union), $\cap$ (intersection), and $-$ (difference) (note that since relations are multisets, the multiset semantics is adopted for the operators, see [UW02], Ch. 5.1).

Semantics. We now provide semantics for the types of databases described thus far.

A *valuation* is a mapping $v : \mathcal{U} \cup \mathcal{N} \to Dom$ such that $U_i \in \mathcal{U}(A)$ implies $v(U_i) \in dom(A)$ and $N_j \in \mathcal{N}(A)$ implies $v(N_j) \in dom(A)$. A valuation v can be applied to a tuple t, relation R, and database I in the obvious way – the result is denoted by $v(t)$, $v(R)$, and $v(I)$, respectively.

Thus, for each attribute A, the application of a valuation replaces each no-information null with a value in $dom(A)$ ($\perp$ allowed) and each unknown null with a value in $dom(A)$ ($\perp$ not allowed) with multiple occurrences of the same null replaced by the same value. The result of applying a valuation is a complete database.

Definition 58. *The set of* completions *of a database I is*

$$comp(I) = \{\ v(I) \mid v \text{ is a valuation } \}.$$

6.3 Partial Information Policies

In this section we introduce *partial information policies* which allow users to make assumptions about missing data in a database, taking into account their own knowledge of how the data was collected, their attitude to risk, and their mission needs.

Definition 59. *Given a database schema DS, a* partial information policy *(PIP) is a mapping $\rho : db(DS) \to 2^{db(DS)}$ s.t. $\rho(I)$ is a non-empty subset of $comp(I)$ for every $I \in db(DS)$.*

Thus, a PIP maps a database to a subset of its completions that we call *preferred completions*.

Example 57. *The completions of the relation in Example 56 are the complete DBs obtained by replacing every unknown value with an actual value. Each user has expressed preferences on which completions are of interest to him. The completions chosen as preferred by user A are those where each unknown under-age enrollment ratio is replaced with a value determined by linear regression; for user B the preferred completions are* *those where unknown under-age enrollment ratios are replaced with values in the range* $[4.81, 16.77]$*; and so forth for the other users.*

Note that the preferred completions chosen by users A, D, E, F (but not B and C) can be *represented* with the data model of Section 6.2, that is, $\forall I \in db(DS) \exists I' \in db(DS)(comp(I') = \rho(I))$. This is so because the policies expressed by users A, D, E, F determine a single actual value for each null value, whereas the policies expressed by users B and C give a set of possible actual values for each null value. The important advantage of this property is that the result of applying a policy can be represented as a database in the same data model of the original database (i.e., the data model of Section 6.2), whereas policies that do not satisfy the property need more expressive and complex data models (e.g., *c-tables* [Gra91, IL84b]). We now present some families of PIPs which enjoy the aforementioned property (the next section defines index structures which allow us to efficiently apply policies with large datasets). In addition, these policies can be used as building blocks to define much more complex policies.

Henceforth, we assume that I is a database and $R \in I$ is a relation over schema S; $A, B \in Att(S)$ and $X, Y, Z \subseteq Att(S)$, with A, B and attributes in Y having numeric domains; μ, ϑ and ν are aggregate operators in $\{MIN, MAX, AVG, MEDIAN, MODE\}$.

Given a tuple $t \in R$, we define (i) the relation $V(t, X, Z) = \{t' \mid t' \in R \wedge t'[X] = t[X] \wedge \forall A_i \in Z\ (t'[A_i] \in dom(A_i))\}$, that is, the multiset of tuples in R having the same X-value as t and a Z-value consisting of values in Dom, and (ii) the relation $V^*(t, X, Z) = \{t' \mid t' \in R \wedge t'[X] = t[X] \wedge \forall A_i \in Z\ (t'[A_i] \in dom(A_i))\}$, that is, the multiset of tuples in R having the same X-value as t and a Z-value consisting of values in $Dom - \{\perp\}$.

Family of Aggregate Policies. $\rho^{agg}(\mu, \nu, A, X)$ is defined as follows. If $t \in R$ and $t[A] \in \mathcal{U}(A)$, then $V = V^*(t, X, \{A\})$, else if $t[A] \in \mathcal{N}(A)$, then $V = V(t, X, \{A\})$. If $\mu\{t'[A] \mid t' \in V\}$ exists, then we say that it is a *candidate value* for $t[A]$. Let I' be the database obtained from I by replacing every occurrence of a null $\eta \in \mathcal{N} \cup \mathcal{U}$ appearing in $\pi_A(R)$ with $\nu\{v_1, \ldots, v_n\}$ (if the latter exists), where the v_i's are the candidate values for η. The preferred completions of this policy are the completions of I'. Note that for each selection of μ, ν, A, X, this single definition defines a different PIP - all belonging to the family of aggregate policies.

Example 58. *For the purpose of illustrating the roles of the different parameters of PIPs, consider the simple relation below.*

Country	*Year*	*GER*	*UER*
Mali	*1996*	*94.67*	*3.84*
Mali	*1997*	*94.83*	U_1
Mali	*1998*	*95.72*	*4.36*
Rwanda	*1996*	*98.84*	*4.67*
Rwanda	*1997*	*103.76*	*5.38*
Rwanda	*1998*	*105.24*	U_1
Senegal	*1997*	*93.14*	*4.52*
Senegal	*1998*	*95.72*	*4.87*
Sudan	*1997*	*102.83*	*5.03*
Sudan	*1998*	*103.76*	*5.12*

Suppose we want to apply the policy $\rho^{agg}(AVG, MAX, UER, \{Country\})$*. This policy looks at missing values under attribute* UER *(third parameter of the policy). When the first occurrence of* U_1 *is considered, a candidate value is computed as follows. Since the last parameter of the policy is* $Country$*, only tuples for Mali are considered and their average (first parameter) UER is a candidate value, i.e.,* 4.1*. Likewise, when the second occurrence of* U_1 *is considered, the average UER for Rwanda, i.e.* 5.025*, is another candidate value. Eventually, the two occurrences of* U_1 *are replaced by* 5.025*, i.e. the* *maximum candidate value (as specified by the second parameter of the policy). If the relation above belongs to a database I, then every occurrence of U_1 elsewhere in I is replaced by* 5.025.

Family of Regression Oriented Policies. $\rho^{reg}(\nu, A, X, Y)$ is defined as follows. If $t \in R$ and $t[A]$ is a null $\eta \in \mathcal{N} \cup \mathcal{U}$, then $\mathcal{D} = \{\langle t'[Y], t'[A]\rangle \mid t' \in V^*(t, X, Y \cup \{A\})\}$. Let f be a model built from $\mathcal{D}$ via linear regression[2], if $\mathcal{D} \neq \emptyset$, where values on Y are the independent variables and values on A are the dependent variables. If $t[Y]$ consists of values in $Dom - \{\perp\}$ only, then $f(t[Y])$ is a candidate value for η. Suppose I' is the database obtained from I by replacing every occurrence of a null $\eta \in \mathcal{N} \cup \mathcal{U}$ in $\pi_A(R)$ with $\nu\{v_1, \ldots, v_n\}$ (if the latter exists), where the v_i's are the candidate values for η. The preferred completions returned by this policy are the completions of I'. Note that this definition defines a very large family of policies - one for each possible way of instantiating ν, A, X, Y.

Example 59. *Consider the relation of Example 58 and suppose we want to apply the policy $\rho^{reg}(AVG, UER, \{Country\}, \{Year\})$. This policy looks at missing values under attribute UER (second parameter of the policy). When the first occurrence of U_1 is considered, a candidate value is computed as follows. As $Country$ is specified as third parameter of the policy, only tuples for Mali are considered. A linear model is built from $\mathcal{D} = \{\langle 1996, 3.84\rangle, \langle 1998, 4.36\rangle\}$. The independent variable of $\mathcal{D}$ is $Year$ (last parameter of the policy). The UER corresponding to* 1997 *given by the linear model is* 4.1, *which is a candidate value. Likewise, when the second occurrence of U_1 is considered, a linear model is built from $\mathcal{D} = \{\langle 1996, 4.67\rangle, \langle 1997, 5.38\rangle\}$ and the candidate value* 6.09

[2] For the sake of simplicity we restrict ourselves to linear regression, but other policies using different regression methods may be defined analogously.

is determined. The two occurrences of U_1 are replaced by the average (first parameter of the policy) of the two candidate values, i.e. 5.095.

Family of Policies Based on Another Attribute. The policy $\rho^{att}(\mu, \vartheta, \nu, A, B, X)$ is defined as follows. If $t \in R$ and $t[A] \in \mathcal{U}(A)$, then $V = V^*(t, X, \{A\})$, else if $t[A] \in \mathcal{N}(A)$, then $V = V(t, X, \{A\})$. If $\beta = \mu\{t'[B] \mid t' \in V\}$ exists, then let $\beta^* = min\{|t'[B] - \beta| : t' \in V\}$[3] and $V' = \{t' \mid t' \in V \wedge |t'[B] - \beta| = \beta^*\}$; we say that $\vartheta\{t'[A] \mid t' \in V'\}$ is a candidate value for $t[A]$. Suppose I' is the database obtained from I by replacing every occurrence of a null $\eta \in \mathcal{N} \cup \mathcal{U}$ appearing in $\pi_A(R)$ with $\nu\{v_1, \ldots, v_n\}$ (if the latter exists), where the v_i's are the candidate values for η. The preferred completions returned by this policy are the completions of I'. This definition also defines a very large family of policies - one for each possible way of instantiating $\mu, \vartheta, \nu, A, B, X$.

Example 60. *Consider again the relation of Example 58 and suppose we want to apply the policy $\rho^{att}(MIN, AVG, MAX, UER, GER, \{Year\})$. This policy looks at missing values under attribute UER (fourth parameter of the policy). A candidate value for the first occurrence of U_1 is determined as follows. Tuples referring to 1997 are considered because the last parameter of the policy is $Year$. Then, the min GER for such tuples is found (this is specified by the first and fifth parameters), i.e.* 93.14*, and the corresponding UER is a candidate value, viz.* 4.52*. Consider now the second occurrence of U_1. Tuples referring to 1998 are considered and the minimum GER is found among those tuples, i.e.* 95.72*. However, there are two tuples having such a value, so there are two corresponding UERs, i.e.* 4.36 *and* 4.87*. The second parameter of the policy states that their average is a candidate, i.e.* 4.615*. Every occurrence of U_1 is replaced by the maximum candidate value, i.e.* 4.615*, as specified by the third parameter of the policy.*

[3] When at least one of x and y is a null, if $x \neq y$, then $|x - y| = \infty$, else $|x - y| = 0$.

When applying a policy to a database, one relation is used to determine how to replace nulls – once replacements have been determined, they are applied to the whole database – thus, different occurrences of the same null are replaced with the same value. Given a database I over schema DS, a relation schema $S \in DS$, and a policy ρ, we use $\rho^S(I)$ to specify that ρ is applied to I and the relation in I over schema S is used to determine the replacements. Once again, note that the preferred completions $\rho^S(I)$ obtained by applying the above policies can be *represented* by a database, i.e., there exists a database I' s.t. $comp(I') = \rho^S(I)$; with a slight abuse of notation we use $\rho^S(I)$ to denote I'.[4]

Example 61. *The policies expressed by Users A, D, E, F in Example 56 can be respectively formulated in the following way:*

1. *a regression policy* $\rho^{reg}(\nu, UER, \{Country\}, \{Year\})$*,*
2. *a regression policy* $\rho^{reg}(\nu, UER, \{Country\}, \{NetODA\})$*,*
3. *an aggregate policy* $\rho^{agg}(MAX, \nu, UER, \{Year\})$*, and*
4. *a policy based on another attribute* $\rho^{att}(AVG, \vartheta, \nu, UER, GER, \{Year\})$*.*

In the PIPs above, ν *determines how multiple candidate values are aggregated and* ϑ *is used as shown in Example 60. Different users may want to apply different PIPs depending on what they believe is more suitable for their purposes – depending on the chosen PIP and the input database, they may get different results.*

The above policies are not exhaustive: they are basic policies that can be combined to obtain more complex ones, e.g., different aggregate policies (on different attributes) can

[4] I' itself need not be complete as nulls may remain in the database in attributes not affected by ρ.

be defined over the same relation schema or an aggregate policy can be combined with a regression oriented policy, and so forth. Furthermore, PIPs can be combined with relational algebra operators allowing users to express even more complex ways of managing their incomplete data – we will deal with relational algebra and PIPs in Section 6.5.

6.4 Efficiently Applying PIPs

In this section, we present index structures to efficiently apply policies and show how they can be incrementally maintained when the database is updated (Section 6.6 presents experimental results showing the index's effectiveness).

Given a PIP ρ of the form $\rho^{agg}(\mu, \nu, A, X)$, $\rho^{reg}(\nu, A, X, Y)$, $\rho^{att}(\mu, \vartheta, \nu, A, B, X)$, we call A the *incomplete attribute* of ρ, denoted as $inc(\rho)$, whereas X is the set of *selection attributes* of ρ, denoted as $sel(\rho)$. Throughout the chapter we will use vector notation to refer to pointers; thus, given a tuple t, $\overrightarrow{t}$ denotes a pointer to t; likewise, given a set c of tuples, $\overrightarrow{c}$ denotes the set of pointers to tuples in c. We start by introducing the notion of *cluster* in the following definition.

Definition 60. *Given a relation R over schema S, and a set of attributes $Z \subseteq Att(S)$, a* cluster *of R w.r.t. Z is a maximal subrelation c of R s.t. $\forall\, t, t' \in c, t[Z] = t'[Z]$. We write $cluster(R, Z)$ to denote the set of clusters of R w.r.t Z; it is the quotient multiset obtained from the identity on Z between tuples in R.*

Example 62. *Consider the relation* ***salary*** *below (throughout this section we use this simple relation as it allows us to clearly illustrate the use of indexes and has all types of incompleteness).*

	$Name$	$Year$	$Salary$
t_1	$John$	2008	$\perp$
t_2	$John$	2009	$60K$
t_3	$John$	2010	U_1
t_4	$Alice$	2009	$70K$
t_5	$Alice$	2010	U_2
t_6	Bob	2009	$60K$
t_7	Bob	2010	$70K$
t_8	$Carl$	2010	N_1

There are four clusters w.r.t. $\{Name\}$*, namely* $c_1 = \{t_1, t_2, t_3\}$*,* $c_2 = \{t_4, t_5\}$*,* $c_3 = \{t_6, t_7\}$ *and* $c_4 = \{t_8\}$*.*

The next example shows the idea behind our index structures.

Example 63. *Suppose we want to apply the policy* $\rho^{agg}(AVG, \nu, Salary, \{Name\})$*, where* ν *is any aggregate operator, to the relation* ***salary*** *of Example 62. To determine how to replace missing salaries, we need to retrieve every cluster in* $cluster($***salary***$, \{Name\})$ *which (i) contains at least one tuple having a missing salary, i.e., a salary in* $\mathcal{U} \cup \mathcal{N}$ *(otherwise there is no need apply the policy to that cluster), and (ii) contains at least one tuple having a non-missing salary (otherwise there is no data from which to infer missing salaries). Clusters satisfying these conditions yield possible candidates – other clusters do not play a role and so can be ignored. In Example 62, we need to retrieve only clusters* c_1 *and* c_2*.*

To leverage this idea, we associate a counter with each cluster to keep track of the number of tuples in the cluster containing standard constants, unknown, no-information, and inapplicable nulls on a specific attribute – the role of such counters will be made clear shortly. Let R be a relation over schema S, $Z \subseteq Att(S)$, and $B \in Att(S)$. Given a cluster $c \in cluster(R, Z)$, we define

- $C_v(c, B) = |\{t \mid t \in c \wedge t[B] \in dom(B)\}|$,
- $C_\perp(c, B) = |\{t \mid t \in c \wedge t[B] = \perp\}|$.
- $C_\mathcal{U}(c, B) = |\{t \mid t \in c \wedge t[B] \in \mathcal{U}(B)\}|$,
- $C_\mathcal{N}(c, B) = |\{t \mid t \in c \wedge t[B] \in \mathcal{N}(B)\}|$,

We now introduce the first data structure that will be used for the efficient application of PIPs.

Definition 61. *Let R and ρ be a relation and a PIP, respectively. Moreover, let $X = sel(\rho)$ and $A = inc(\rho)$. A* cluster table *for R and ρ is defined as follows:*

$$\begin{aligned} &ct(R, \rho) = \\ &\quad \{\langle t[X], \overrightarrow{c}, C_v(c, A), C_\perp(c, A), C_\mathcal{U}(c, A), C_\mathcal{N}(c, A)\rangle \\ &\qquad s.t.\ c \in cluster(R, X) \wedge t \in c\} \end{aligned}$$

Example 64. *The cluster table T for the relation* **salary** *of Example 62 and the policy $\rho^{agg}(AVG, \nu, Salary, \{Name\})$, where ν is an arbitrary aggregate is:*

	$t[\{Name\}]$	$\overrightarrow{c}$	C_v	$C_\perp$	$C_\mathcal{U}$	$C_\mathcal{N}$
s_1	*John*	$\{\overrightarrow{t_1}, \overrightarrow{t_2}, \overrightarrow{t_3}\}$	1	1	1	0
s_2	*Alice*	$\{\overrightarrow{t_4}, \overrightarrow{t_5}\}$	1	0	1	0
s_3	*Bob*	$\{\overrightarrow{t_6}, \overrightarrow{t_7}\}$	2	0	0	0
s_4	*Carl*	$\{\overrightarrow{t_8}\}$	0	0	0	1

where C_v stands for $C_v(c, Salary)$, $C_\perp$ stands for $C_\perp(c, Salary)$, and so forth.

The counters associated with each cluster in a cluster table determine whether a policy needs the cluster to determine candidate values. For instance, the PIP in Example 64 has to look at those clusters having $C_{\mathcal{U}}(c, Salary) + C_{\mathcal{N}}(c, Salary) > 0$ (i.e., having some missing salaries) and $C_v(c, Salary) > 0$ (i.e., having some salaries to be exploited for inferring the missing ones). Different conditions determine whether a given PIP needs to consider a given cluster.

Definition 62. *Suppose R is a relation, ρ is a PIP, and c is a cluster in $cluster(R, sel(\rho))$.*
1) If ρ is an aggregate policy $\rho^{agg}(\mu, \nu, A, X)$ (ν being any aggregate operator), then

- *if $\mu \in \{MAX, MIN, AVG, MEDIAN\}$ and $C_{\mathcal{U}}(c, A)+C_{\mathcal{N}}(c, A) > 0 \wedge C_v(c, A) > 0$, then we say that c is* relevant *w.r.t. ρ;*

- *if $\mu = MODE$ and $\big((C_v(c, A) > 0 \wedge C_{\mathcal{U}}(c, A) > 0) \vee (C_{\mathcal{N}}(c, A) > 0 \wedge C_v(c, A) + C_\perp(c, A) > 0)\big)$, then we say that c is* relevant *w.r.t. ρ.*

2) If ρ is a regression oriented policy $\rho^{reg}(\nu, A, X, Y)$ (ν being any aggregate operator) and $C_{\mathcal{U}}(c, A) + C_{\mathcal{N}}(c, A) > 0 \wedge C_v(c, A) > 0$, then we say that c is relevant *w.r.t. ρ.*
3) If ρ is a policy based on another attribute $\rho^{att}(\mu, \vartheta, \nu, A, B, X)$ (μ, ϑ and ν are any aggregate operators) and $\big((C_v(c, A) > 0 \wedge C_{\mathcal{U}}(c, A) > 0) \vee (C_{\mathcal{N}}(c, A) > 0 \wedge C_v(c, A) + C_\perp(c, A) > 0)\big)$, then c is relevant *w.r.t. ρ.*

The counters associated with each cluster in a cluster table allow us to determine whether a cluster is relevant or not without scanning the entire cluster. Furthermore, as we will discuss later, when the database is modified (i.e., tuple insertions, deletions, or updates occur) such counters allow us to determine whether the "relevance" of a cluster changes or not without scanning it.

For a cluster table T we maintain an additional data structure that allows us to retrieve the tuples of T which refer to relevant clusters.

Definition 63. *Let $T = ct(R, \rho)$ be the cluster table for a relation R and a PIP ρ. Then,*

$$\begin{aligned} & Relevant(T, \rho) = \\ & \quad \{\langle t[X], \overrightarrow{s} \rangle \mid s = \langle t[X], \overrightarrow{c}, C_v, C_\perp, C_{\mathcal{U}}, C_{\mathcal{N}} \rangle \in T \\ & \quad \quad \wedge\ c \textit{ is relevant w.r.t. } \rho\} \end{aligned}$$

Example 65. *Consider the cluster table T and the PIP of Example 64; $Relevant(T, \rho)$ is as follows:*

$t[\{Name\}]$	$\overrightarrow{s}$
$John$	$\overrightarrow{s_1}$
$Alice$	$\overrightarrow{s_2}$

The following proposition states the complexity of building a cluster table $T = ct(R, \rho)$ and $Relevant(T, \rho)$ for a relation R and a PIP ρ.

Proposition 28. Let R and ρ be a relation and a PIP, respectively. Independent of ρ the worst-case time complexity of building $T = ct(R, \rho)$ and $Relevant(T, \rho)$ is $O(|R| \cdot log|R|)$.

A cluster table $T = ct(R, \rho)$ and $Relevant(T, \rho)$ are maintained for each policy ρ. Note that policies having the same $sel(\rho)$ and $inc(\rho)$ can share the same cluster table T; moreover, if the the criterion that determines whether a cluster is relevant or not is the same (see Definition 62), they can also share the same $Relevant(T, \rho)$.

Recall that when a policy determines a value c which has to replace a null η, then every occurrence of η in the database has to be replaced with c. Thus, whenever a value has been determined for a null, we need to retrieve all those tuples containing that null. To this end, we maintain the data structure defined in the definition below. Given a null $\eta \in \mathcal{N} \cup \mathcal{U}$ and a database I, I_η denotes the set of tuples in I containing η.

Definition 64. *Given a database I, we define*

$$Null(I) = \{\langle \eta, \overrightarrow{I_\eta} \rangle \mid \eta \in \mathcal{N} \cup \mathcal{U} \wedge \overrightarrow{I_\eta} \neq \emptyset\}$$

Clearly, $Null(I)$ is shared by all the policies.

Proposition 29. Given a database I, the worst-case time complexity of building $Null(I)$ is $O(|I| \cdot (log\, N_{null} + log|I_{\eta_{max}}|))$, where $|I|$ is the number of tuples in I, N_{null} is the number of distinct nulls in $\mathcal{N} \cup \mathcal{U}$ appearing in I, and $I_{\eta_{max}}$ is the I_η with maximum cardinality.

In the rest of this section we show how to update a cluster table T, $Relevant(T, \rho)$ and $Null(I)$ when tuples are inserted, deleted, or updated. We also show how to apply a PIP exploiting the data structures presented thus far and introduce further optimizations that can be applied for specific policies.

6.4.1 Tuple insertions

Figure 6.1 reports an algorithm to update a cluster table T, $Relevant(T, \rho)$ and $Null(I)$ after a tuple t is inserted. The algorithm first updates $Null(I)$ (lines 1–3). After that, if there already exists a cluster c for t, then $\overrightarrow{t}$ is added to $\overrightarrow{c}$ (line 5), the counters associated with c are properly updated (line 6), and if c becomes a relevant cluster, then a tuple for c is added to $Relevant(T, \rho)$ (line 7). Note that in order to determine whether a cluster is relevant or not, it suffices to check its associated counters instead of scanning the entire cluster. If there does not exist a cluster for t, then a new tuple for it is added to T (lines 8–11) – in this case the cluster is certainly non-relevant.

```
     Algorithm CT-insert
     Input: A relation $R \in I$, a PIP $\rho$, cluster table $T = ct(R, \rho)$,
            $Relevant(T, \rho)$, $Null(I)$, and a new tuple $t$
            ($X = sel(\rho)$ and $A = inc(\rho)$)
1    For each $\eta \in \mathcal{N} \cup \mathcal{U}$ appearing in $t$
2      If $\exists \langle \eta, \overrightarrow{I_\eta} \rangle \in Null(I)$ then Add $\overrightarrow{t}$ to $\overrightarrow{I_\eta}$
3      else Add $\langle \eta, \{\overrightarrow{t}\} \rangle$ to $Null(I)$
4    If $\exists s = \langle t[X], \overrightarrow{c}, C_v, C_\perp, C_\mathcal{U}, C_\mathcal{N} \rangle \in T$ then
5      Add $\overrightarrow{t}$ to $\overrightarrow{c}$
6      Update one of $C_v$, $C_\perp$, $C_\mathcal{U}$, $C_\mathcal{N}$ according to $t[A]$
7      If $c$ has become relevant then Add $\langle t[X], \overrightarrow{s} \rangle$ to $Relevant(T, \rho)$
8    else If $t[A] \in dom(A)$ then Add $\langle t[X], \{\overrightarrow{t}\}, 1, 0, 0, 0 \rangle$ to $T$
9        else If $t[A] = \perp$ then Add $\langle t[X], \{\overrightarrow{t}\}, 0, 1, 0, 0 \rangle$ to $T$
10       else If $t[A] \in \mathcal{U}(A)$ then Add $\langle t[X], \{\overrightarrow{t}\}, 0, 0, 1, 0 \rangle$ to $T$
11       else Add $\langle t[X], \{\overrightarrow{t}\}, 0, 0, 0, 1 \rangle$ to $T$
```

Figure 6.1: Updating index structures after a tuple insertion.

Example 66. *Suppose we add tuple $t = \langle Bob, 2011, U_4 \rangle$ to the relation **salary** of Example 62. First, a new tuple $\langle U_4, \{\overrightarrow{t}\} \rangle$ is added to $Null(\{$**salary**$\})$. As there is already a cluster for Bob, s_3 is retrieved from T (see the cluster table T in Example 64), $\overrightarrow{t}$ is added to the set of pointers of the cluster and $C_\mathcal{U}$ is incremented by one, i.e., s_3 becomes $\langle Bob, \{\overrightarrow{t_6}, \overrightarrow{t_7}, \overrightarrow{t}\}, 2, 0, 1, 0 \rangle$. As the cluster is relevant w.r.t. the policy of Example 64, $\langle Bob, \overrightarrow{s_3} \rangle$ is added to $Relevant(T, \rho)$.*

The following two propositions state the correctness and the complexity of Algorithm CT-insert. With a slight abuse of notation, we use $I \cup \{t\}$ to denote the database obtained by adding t to R, R being a relation of I.

Proposition 30. Let R be a relation of a database I, ρ a PIP, and t a tuple. Algorithm CT-insert computes $T' = ct(R \cup \{t\}, \rho)$, $Relevant(T', \rho)$ and $Null(I \cup \{t\})$.

Proposition 31. The worst-case time complexity of Algorithm CT-insert is $O(log|Null(I)| + log|I_{\eta_{max}}| + log|T| + log|c_{max}|)$, where c_{max} is the cluster with maximum cardinality and $I_{\eta_{max}}$ is the set of tuple pointers in $Null(I)$ with maximum cardinality.

6.4.2 Tuple deletions

Figure 6.2 presents an algorithm to update a cluster table T, $Relevant(T, \rho)$ and $Null(I)$ when deleting a tuple t.

```
Algorithm CT-delete
Input: A relation $R \in I$, a PIP $\rho$, cluster table $T = ct(R, \rho)$,
       $Relevant(T, \rho)$, $Null(I)$, and a tuple $t$
       ($X = sel(\rho)$ and $A = inc(\rho)$)

For each $\eta \in \mathcal{N} \cup \mathcal{U}$ appearing in $t$
   Get $\langle \eta, \overrightarrow{I_\eta} \rangle$ from $Null(I)$
   Delete $\overrightarrow{t}$ from $\overrightarrow{I_\eta}$
   If $\overrightarrow{I_\eta} = \emptyset$ then Delete $\langle \eta, \overrightarrow{I_\eta} \rangle$ from $Null(I)$
Get $s = \langle t[X], \overrightarrow{c}, C_v, C_\perp, C_\mathcal{U}, C_\mathcal{N} \rangle$ from $T$
Delete $\overrightarrow{t}$ from $\overrightarrow{c}$
Update one of $C_v$, $C_\perp$, $C_\mathcal{U}$, $C_\mathcal{N}$ according to $t[A]$
   If $c$ has become non-relevant then
      Delete $\langle t[X], \overrightarrow{s} \rangle$ from $Relevant(T, \rho)$
If $\overrightarrow{c} = \emptyset$ then Delete $s$ from $T$
```

Figure 6.2: Updating index structures after a tuple deletion.

Example 67. *Suppose we delete t_4 from the relation **salary** of Example 62. Then, no changes are made to $Null(\{$**salary**$\})$. s_2 is retrieved from T (see the cluster table T in Example 64), $\overrightarrow{t_4}$ is deleted from the set of pointers of the cluster and C_v is decremented by one, that is, s_2 becomes $\langle Alice, \{\overrightarrow{t_5}\}, 0, 0, 1, 0 \rangle$. As the cluster is not relevant anymore, $\langle Alice, \overrightarrow{s_2} \rangle$ is deleted from $Relevant(T, \rho)$.*

The propositions below state the correctness and the complexity of Algorithm CT-delete. With a slight abuse of notation, we use $I - \{t\}$ to denote the database obtained by deleting t from R, R being a relation of I.

Proposition 32. Let R be a relation of a database I, ρ a PIP, and t a tuple in R. Algorithm CT-delete computes $T' = ct(R - \{t\}, \rho)$, $Relevant(T', \rho)$, and $Null(I - \{t\})$.

Proposition 33. The worst-case time complexity of Algorithm CT-delete is the same as for Algorithm CT-insert.

6.4.3 Tuple updates

An algorithm for updating a cluster table T, $Relevant(T, \rho)$ and $Null(I)$ after a tuple t is updated to t' can be simply defined by first calling CT-delete with t as parameter and then calling CT-insert with t' as parameter. We call this algorithm CT-update. The following two propositions state the correctness and the complexity of Algorithm CT-update. With a slight abuse of notation, we use $I - \{t\} \cup \{t'\}$ to denote the database obtained by updating $t \in R$ into t', R being a relation of I.

Proposition 34. Let R be a relation of a database I, ρ a PIP, t a tuple in R, and t' a tuple. Algorithm CT-update computes $T' = ct(R - \{t\} \cup \{t'\}, \rho)$, $Relevant(T', \rho)$ and $Null(I - \{t\} \cup \{t'\})$.

Proposition 35. The worst-case time complexity of Algorithm CT-update is the same as for Algorithm CT-insert.

Algorithm CT-update can be optimized when t and t' belong to same cluster; we omit the optimized algorithm and illustrate the basic intuition below. Consider the relation **salary** of Example 62 and suppose we modify $t_4 = \langle Alice, 2009, 70K\rangle$ to $t'_4 = \langle Alice, 2009, 80K\rangle$. If we first execute Algorithm CT-delete with t_4, then its cluster becomes irrelevant and the corresponding tuple is deleted from $Relevant(T, \rho)$. When we execute CT-insert with t'_4, $Alice$'s cluster becomes relevant again and a tuple for it is inserted into $Relevant(T, \rho)$. As another example, consider $t_8 = \langle Carl, 2010, N_1\rangle$ and suppose Carl's salary is modified. By executing CT-delete and CT-insert, s_4 is first deleted from T (see the cluster table in Example 64) and then it is added again.

Deleting from and inserting into $Relevant(T, \rho)$ or T can be avoided if we first check whether t and t' belong to the same cluster and if so, then we do not call CT-delete and CT-insert, but directly update $Relevant(T, \rho)$ and T according to $t[A]$ and $t[A']$ (in addition, $Null(I)$ is updated according to the null values in t and t').

6.4.4 Applying PIPs

Figure 6.3 shows the CT-ApplyPIP algorithm to apply a PIP on top of our data structures. CT-ApplyPIP first retrieves the relevant clusters so that only a subrelation R' of R has to be considered in order to determine how to replace null values (lines 1–4). The policy ρ then tries to determine a value for each null appearing in R' on attribute A (lines 5–6) – this depends on the adopted policy and is accomplished as described in Section 6.3. If a value v for a null η has been determined, then every occurrence of η in the database is replaced with v (lines 7–10). It is worth noting that when a null is replaced with a value, then CT-update is executed.

```
Algorithm CT-ApplyPIP
Input: A relation $R \in I$, a PIP $\rho$, $T = ct(R, \rho)$,
       $Relevant(T, \rho)$ and $Null(I)$
       ($X = sel(\rho)$ and $A = inc(\rho)$)

$R' = \emptyset$
For each $\langle t[X], \overrightarrow{s} \rangle \in Relevant(T, \rho)$
    Get $s = \langle t[X], \overrightarrow{c}, C_v, C_\perp, C_\mathcal{U}, C_\mathcal{N} \rangle$ from $T$
    $R' = R' \cup c$
For each $\eta \in \mathcal{N} \cup \mathcal{U}$ appearing in $R'$ on $A$
    Determine a value $v$ for $\eta$ according to $\rho$
    If $v$ exists then
        Get $\langle \eta, \overrightarrow{I_\eta} \rangle$ from $Null(I)$
        For each $\overrightarrow{t} \in \overrightarrow{I_\eta}$
            Replace every occurrence of $\eta$ in $t$ with $v$
```

Figure 6.3: Applying a PIP

The following two propositions state the correctness and the complexity of Algorithm CT-ApplyPIP.

Proposition 36. Let I be a database, $R \in I$ a relation over schema S, and ρ a PIP. Algorithm CT-ApplyPIP correctly computes $I' = \rho^S(I)$, $T' = ct(R', \rho)$, $Relevant(T', \rho)$ and $Null(I')$, where $R' \in I'$ is the relation over schema S.

Proposition 37. The worst-case time complexity of Algorithm CT-ApplyPIP is $O(|R'| \cdot (cost_\rho(R') + log|Null(I)| + |I_{\eta_{max}}| \cdot cost_{\text{CT-update}}))$, where R' is the union of the relevant clusters of R (w.r.t. ρ), $cost_\rho$ is the cost of determining a value for a null according to policy ρ, $I_{\eta_{max}}$ is the set of tuple pointers in $Null(I)$ with maximum cardinality, and $cost_{\text{CT-update}}$ is the cost of updating a tuple (see Proposition 35).

Basically, applying a policy consists of determining how to replace every null appearing for the incomplete attribute and then replacing every occurrence of it in the database (note that the former step needs the clusters to be identified). When applying a policy, the data structures introduced in this section have the following benefits. *1)* The relation from which candidate values are determined does not have to be scanned to identify its clusters, as they can be efficiently retrieved from the cluster table. *2)* By looking at $Relevant(T, \rho)$, only those clusters from which candidate values can be determined (i.e., the relevant clusters) are considered when applying a policy, thus avoiding looking at those tuples in the relation that can be disregarded. *3)* Tuples containing nulls that have to be replaced can be retrieved from $Null(I)$ without scanning the whole database. *4)* When the database is modified because of tuple insertions, deletions or updates, our data structures can be efficiently updated.

Experiments (cf. Section 6.6) show that these indexes yield significant performance gains on large datasets.

6.4.5 Optimizations

The data structures and the algorithms presented above can be optimized as illustrated in the following example.

Example 68. *Consider the relation* ***salary*** *from Example 62. Let ν is any aggregate operator, then assuming a policy $\rho^{agg}(AVG, \nu, Salary,$ $\{Name\})$, the corresponding cluster table T and $Relevant(T, \rho)$ are reported in Examples 64 and 65, respectively. For each tuple $\langle t[X], \overrightarrow{c}, C_v, C_\perp, C_\mathcal{U}, C_\mathcal{N}\rangle$ in T we might keep the average of the salaries (in general, the average of A-values, A being the incomplete attribute of the PIP) of the tuples in c. Thus, when the policy is applied, the average salary of each cluster can be obtained without scanning the entire cluster.*

The average salary can be inexpensively computed when the cluster table is first built. In addition, this value can be easily updated when the relation ***salary*** *is updated. For instance, when a new tuple t is inserted, if $t[Salary] \notin dom(Salary)$, then nothing has to be done. If $t[Salary] \in dom(Salary)$, then the new average salary is computed as*

$avg_{new} = \frac{avg_{old} \cdot C_v + t[Salary]}{C_v + 1}$*, where* avg_{old} *is the old average salary and* C_v *is the number of salaries in* t*'s cluster (before inserting* t*). Likewise, the average salary can be updated when tuples are deleted or updated.*

The optimization in the example above can be applied to other policies as well, the basic idea is to associate each tuple in a cluster table with a pre-computed value (or a set of pre-computed values) which turns out to be useful when determining candidate values – such a value is then incrementally updated when the database is modified.

6.5 Relational Algebra and PIPs

In this section, we study when applying a policy before a relational algebra operator gives the same result as applying it after. This can be exploited for query optimization purposes. Note that PIPs cannot be expressed using the relational algebra because PIPs modify the database by replacing null values; while the relational algebra operators cannot modify the database. Throughout this section, a policy is either an aggregate policy, or a regression policy, or a policy based on another attribute. We adopt the SQL semantics [SQL03] for the evaluation of relational algebra operators.

We now define the database obtained after applying a relational algebra operator. The database obtained by applying projection to a relation R is defined as the database obtained by replacing R with its projection. More formally, consider a database I over schema DS, and a relation $R \in I$ over schema $S \in DS$. In addition, let $Z \subseteq \mathit{Att}(S)$. We define $\pi_Z^S(I) = (I \cup \{\pi_Z(R)\}) - \{R\}$. Thus, the notation $\pi_Z^S(I)$ means that the relation R in I over schema S is replaced by $\pi_Z(R)$. Likewise, the database obtained as the result of performing the cartesian product of two relations R_1 and R_2 is defined as the database obtained by replacing R_1 and R_2 with $R_1 \times R_2$. Stated more formally, consider a database I over schema DS, and two relations $R_1, R_2 \in I$ over schemas $S_1, S_2 \in DS$. We define $\times^{S_1,S_2}(I) = (I \cup \{R_1 \times R_2\}) - \{R_1, R_2\}$. The result databases for the other relational algebra operators are defined similarly and analogous notations will be used for them.

6.5.1 Projection and PIPs

We consider both the case where projection returns a set and the case where a multiset is returned. For notational convenience, we use π^m_Z to denote the projection operator which returns a multiset, whereas π_Z denotes the projection operator that returns a set. In order for a PIP to make sense after projection, we assume that the attributes on which the projection is performed include the attributes appearing in the policy. The following proposition considers the projection operator that returns a set and provides sufficient conditions under which applying a policy before or after projection gives the same result.

Proposition 38. Suppose I is a database over schema DS, $R \in I$ is a relation over schema $S \in DS$, $Z \subseteq \mathit{Att}(S)$, and ρ is a policy. Moreover, let S' denote the schema of $\pi_Z(R)$.

1. If ρ is an aggregate policy $\rho^{agg}(\mu, \nu, A, X)$, then

   (a) if $\mu \in \{\mathit{MAX}, \mathit{MIN}\}$, then $\rho^{S'}(\pi^S_Z(I)) = \pi^S_Z(\rho^S(I))$;

   (b) if $\mu \in \{\mathit{AVG}, \mathit{MEDIAN}, \mathit{MODE}\}$, then $\rho^{S'}(\pi^S_Z(I)) = \pi^S_Z(\rho^S(I))$ if $\pi_Z(C) = \pi^m_Z(C)$, where $C = \bigcup_{c \in cluster(R,X) \wedge c \textit{ is relevant w.r.t. } \rho} c$.

2. If ρ is a policy based on another attribute $\rho^{att}(\mu, \vartheta, \nu, A, B, X)$, then

(a) if $\mu, \vartheta \in \{MAX, MIN\}$, then $\rho^{S'}(\pi^S_Z(I)) = \pi^S_Z(\rho^S(I))$;

(b) otherwise $\rho^{S'}(\pi^S_Z(I)) = \pi^S_Z(\rho^S(I))$ if $\pi_Z(C) = \pi^m_Z(C)$,

where $C = \bigcup_{c \in cluster(R,X) \wedge c \text{ } is \text{ } relevant \text{ } w.r.t. \text{ } \rho} c$.

3. If ρ is a regression oriented policy, then $\rho^{S'}(\pi^S_Z(I)) = \pi^S_Z(\rho^S(I))$.

Thus, applying a PIP before or after projection does not always give the same result when we consider aggregate policies or policies based on another attribute using one of the operators AVG, $MEDIAN$, $MODE$. Here the point is that projection loses duplicates; while this makes no difference when MAX or MIN are used, such a loss may change the result of the other aggregate operators. When a regression policy is applied, the loss of duplicates does not change the set of data used to build the regression model. The following example shows that the sufficient conditions stated above are not necessary conditions.

Example 69. *Consider the database I consisting of the relation R below and let S denote the schema of R.*

A	B	C	D
U_1	b	1	d_1
2	b	2	d_2
2	b	2	d_3

Let ρ be an aggregate policy $\rho^{agg}(\mu, \nu, A, \{B\})$ or a policy based on another attribute $\rho^{att}(\mu, \vartheta, \nu, A, C, \{B\})$, with $\mu \in \{AVG, MEDIAN, MODE\}$ and ϑ, ν arbitrary aggregate operators. Moreover, let S' denote the schema of $\pi_{ABC}(R)$. We have that $\rho^{S'}(\pi^S_{ABC}(I)) = \pi^S_{ABC}(\rho^S(I))$ even though $\pi_{ABC}(C) \neq \pi^m_{ABC}(C)$ – here C is defined as in Proposition 38.

If the projection operator which returns a multiset instead of a set is used, then the two orders in which a policy can be applied always give the same result.

Proposition 39. Suppose I is a database over schema DS, $R \in I$ is a relation over schema $S \in DS$, $Z \subseteq Att(S)$, and ρ is a policy. Moreover, let S' denote the schema of $\pi_Z^m(R)$. If projection returns a multiset, then $\rho^{S'}(\pi_Z^S(I)) = \pi_Z^S(\rho^S(I))$.

6.5.2 Selection and PIPs

Applying a PIP before or after selection yields different results in very simple cases as shown in the example below. The intuitive reason is that the two orders give different results when the selection applied first does not keep tuples which affect the application of the policy.

Example 70. *Consider the database I consisting of the relation R below and let S denote the schema of R.*

A	B	C
U_1	b	1
2	b	2

Consider $\rho^S(I)$, where ρ is one of the following policies:

- $\rho^{agg}(\mu, \nu, A, \{B\})$. *This policy replaces U_1 with 2, for any aggregate operators μ and ν.*
- $\rho^{reg}(\nu, A, \{B\}, \{C\})$. *By applying linear regression, U_1 is replaced by 1 for any aggregate operator ν.*

- $\rho^{att}(\mu, \vartheta, \nu, A, C, \{B\})$. *For any aggregate operators* μ, ϑ *and* ν, U_1 *is replaced with* 2.

For any of the policies above, $\rho^S(\sigma^S_{C=1}(I)) \neq \sigma^S_{C=1}(\rho^S(I))$. *In each case,* $\rho^S(I)$ *is obtained by replacing* U_1 *with a value which is determined using the second tuple in* R *(this happens because the two tuples in* R *have the same* B*-value). Clearly,* $\sigma^S_{C=1}(\rho^S(I))$ *returns the first tuple in* R *where* U_1 *has been replaced with an actual value. On the other hand, when selection is first applied, the second tuple in* R *is deleted and then the subsequent application of a policy has no effect because there is no data to infer an actual value for* U_1*. Thus,* $\rho^S(\sigma^S_{C=1}(I))$ *gives exactly the first tuple in* R *leaving* U_1 *as is. Note that neither of the two results contains the other.*

6.5.3 Cartesian Product and PIPs

In the following proposition we identify different ways in which cartesian product and PIPs interact one another.

Proposition 40. Suppose I is a database over schema DS, $R_1, R_2 \in I$ are relations over schemas $S_1, S_2 \in DS$, and ρ_1, ρ_2 are policies for the former and latter relations, respectively. Furthermore, let S' denote the schema of $R_1 \times R_2$, and W_1, W_2 be the attributes appearing in ρ_1 and ρ_2, respectively. Then,

1) $\rho_1^{S'}(\times^{S_1,S_2}(I)) = \times^{S_1,S_2}(\rho_1^{S_1}(I))$.

2) $\rho_2^{S'}(\times^{S_1,S_2}(I)) = \times^{S_1,S_2}(\rho_2^{S_2}(I))$.

3) $\rho_2^{S'}(\rho_1^{S'}(\times^{S_1,S_2}(I))) = \times^{S_1,S_2}(\rho_2^{S_2}(\rho_1^{S_1}(I)))$.

4) $\rho_1^{S'}(\rho_2^{S'}(\times^{S_1,S_2}(I))) = \times^{S_1,S_2}(\rho_1^{S_1}(\rho_2^{S_2}(I)))$.

5) If $\pi_{W_1}(R_1)$ and $\pi_{W_2}(R_2)$ do not have nulls in common, then $\rho_1^{S'}(\rho_2^{S'}(\times^{S_1,S_2}(I))) = \rho_2^{S'}(\rho_1^{S'}(\times^{S_1,S_2}(I)))$.

The fifth item above provides a sufficient condition to guarantee that the two different orders in which ρ_1 and ρ_2 are applied after performing the cartesian product give the same result. The following example shows that this is not a necessary condition.

Example 71. *Consider the database I consisting of the following two relations R_1 and R_2 (whose schemas are denoted by S_1 and S_2, respectively):*

A	B	C
U_1	b	1
2	b	1

D	E	F
U_1	e	1
2	e	1

Let ρ_1 be either of this policies, $\rho^{agg}(\mu_1, \nu_1, A, \{B\})$, or $\rho^{reg}(\nu_1, A, \{B\}, \{C\})$, or $\rho^{att}(\mu_1, \vartheta_1, \nu_1, A, C, \{B\})$; and ρ_2 be either $\rho^{agg}(\mu_2, \nu_2, D, \{E\})$ or $\rho^{reg}(\nu_2, D, \{E\}, \{F\})$ or $\rho^{att}(\mu_2, \vartheta_2, \nu_2, D, F, \{D\})$. Let S' denote the schema of $R_1 \times R_2$. For any choice of the aggregate operators, even though $\pi_{W_1}(R_1)$ and $\pi_{W_2}(R_2)$ have nulls in common (here W_1 and W_2 are defined as in Proposition 40), the following holds $\rho_1^{S'}(\rho_2^{S'}(\times^{S_1,S_2}(I))) = \rho_2^{S'}(\rho_1^{S'}(\times^{S_1,S_2}(I)))$.

6.5.4 Join and PIPs

The join $R_1 \bowtie_\varphi R_2$ of R_1, R_2 can be rewritten as the expression $\sigma_\theta(\sigma_{\theta_1}(R_1) \times \sigma_{\theta_2}(R_2))$, for some $\theta, \theta_1, \theta_2$. This equivalence can be effectively exploited.

Corollary 6. *Suppose I is a database over schema DS, $R_1, R_2 \in I$ are relations over schemas $S_1, S_2 \in DS$, and ρ_1, ρ_2 are policies for the former and latter relation, respectively. Let $R_1 \bowtie_\varphi R_2 = \sigma_\theta(\sigma_{\theta_1}(R_1) \times \sigma_{\theta_2}(R_2))$ for some $\theta, \theta_1, \theta_2$. Furthermore, let S' denote the schema of $\sigma_{\theta_1}(R_1) \times \sigma_{\theta_2}(R_2)$, and W_1, W_2 be the attributes appearing in ρ_1 and ρ_2, respectively. Then,*

1. *$\sigma^{S'}_{\theta}(\rho^{S'}_{1}(\times^{S_1,S_2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I))))) = \sigma^{S'}_{\theta}(\times^{S_1,S_2}(\rho^{S_1}_{1}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I)))))$.*

2. *$\sigma^{S'}_{\theta}(\rho^{S'}_{2}(\times^{S_1,S_2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I))))) = \sigma^{S'}_{\theta}(\times^{S_1,S_2}(\rho^{S_2}_{2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I)))))$.*

3. *$\sigma^{S'}_{\theta}(\rho^{S'}_{2}(\rho^{S'}_{1}(\times^{S_1,S_2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I)))))) = \sigma^{S'}_{\theta}(\times^{S_1,S_2}(\rho^{S_2}_{2}(\rho^{S_1}_{1}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I))))))$.*

4. *$\sigma^{S'}_{\theta}(\rho^{S'}_{1}(\rho^{S'}_{2}(\times^{S_1,S_2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I)))))) = \sigma^{S'}_{\theta}(\times^{S_1,S_2}(\rho^{S_1}_{1}(\rho^{S_2}_{2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I))))))$.*

5. *If $\pi_{W_1}(R_1)$ and $\pi_{W_2}(R_2)$ do not have nulls in common, then*

   $\sigma^{S'}_{\theta}(\rho^{S'}_{1}(\rho^{S'}_{2}(\times^{S_1,S_2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I)))))) = \sigma^{S'}_{\theta}(\rho^{S'}_{2}(\rho^{S'}_{1}(\times^{S_1,S_2}(\sigma^{S_2}_{\theta_2}(\sigma^{S_1}_{\theta_1}(I))))))$.

6.5.5 Union and PIPs

We provide a sufficient condition under which the policy first and policy last strategies return the same result.

Proposition 41. Suppose I is a database over schema DS, $R_1, R_2 \in I$ are relations over schemas $S_1 = r_1(A_1, \ldots, A_n), S_2 = r_2(A_1, \ldots, A_n) \in DS$, and ρ is a PIP. Furthermore, let S' denote the schema of $R_1 \cup R_2$, W the attributes appearing in ρ, and $X = sel(\rho)$. If $\pi_X(R_1) \cap \pi_X(R_2) = \emptyset$ and $\pi_W(R_1)$, $\pi_W(R_2)$ do not have nulls in common, then $\rho^{S'}(\cup^{S_1,S_2}(I)) = \cup^{S_1,S_2}(\rho^{S_2}(\rho^{S_1}(I))) = \cup^{S_1,S_2}(\rho^{S_1}(\rho^{S_2}(I)))$.

The next example shows that the condition in the previous proposition is not a necessary condition.

Example 72. *Consider the database $I = \{R_1, R_2\}$ where R_1 and R_2 are shown below. Let $S_1 = r_1(A, B, C, D)$ and $S_2 = r_2(A, B, C, D)$ denote their schemas, respectively.*

A	B	C	D
U_1	b	1	d_1
2	b	1	d_2

A	B	C	D
U_2	b	1	d_3
2	b	1	d_4

Let ρ be either $\rho^{agg}(\mu, \nu, A, \{B\})$, $\rho^{reg}(\nu, A, \{B\}, \{C\})$, or $\rho^{att}(\mu, \vartheta, \nu, A, C, \{B\})$. For any aggregate operators μ, ϑ and ν, we have that $\rho^{S'}(\cup^{S_1,S_2}(I)) = \cup^{S_1,S_2}(\rho^{S_2}(\rho^{S_1}(I))) = \cup^{S_1,S_2}(\rho^{S_1}(\rho^{S_2}(I)))$ even though $\pi_B(R) \cap \pi_B(R') \neq \emptyset$ (S' denotes the schema of $R_1 \cup R_2$).

6.5.6 Difference and PIPs

As we show in the example below, the different orders in which a policy can be combined with the difference operator yield different results in very simple cases; the reason is similar to the one given for selection.

Example 73. *Consider the database I consisting of the relations R_1 and R_2 below, and let $S_1 = r_1(A, B, C, D)$ and $S_2 = r_2(A, B, C, D)$ denote their schemas, respectively.*

A	B	C	D
U_1	b	1	d_1
2	b	1	d_2

A	B	C	D
2	b	1	d_2

Suppose we compute $\rho^{S_1}(I)$, where ρ is any of the following policies.

1. $\rho^{agg}(\mu, \nu, A, \{B\})$. *This policy replaces U_1 with 2 for any aggregate operators μ and ν.*

2. $\rho^{reg}(\nu, A, \{B\}, \{C\})$. *By applying linear regression, U_1 is replaced by 2 for any aggregate operator ν.*

3. $\rho^{att}(\mu, \vartheta, \nu, A, C, \{B\})$. *$U_1$ is replaced with 2 for any aggregate operators μ, ϑ, ν.*

Thus, for any of the policies above, $\rho^{S_1}(I)$ replaces U_1 with a value determined using the second tuple in R_1 (this is because the two tuples in R_1 have the same B-value). Clearly, $-^{S_1,S_2}(\rho^{S_1}(I))$ returns only the first tuple in R_1 where U_1 has been replaced with *an actual value. However, if the difference operator is performed before applying ρ, then the first tuple in R_1 is returned and the application of a policy afterwards has no effect because there are no tuples that can be used to determine al value for U_1. Hence, we get different results depending on whether we apply the policy before or after the difference operator. Moreover neither result includes the other.*

6.5.7 Intersection and PIPs

As in the case of difference, applying a policy before or after intersection leads to different results in simple cases.

Example 74. *Consider a database I consisting of the relations R_1 and R_2 below and let $S_1 = r_1(A, B, C, D)$ and $S_2 = r_2(A, B, C, D)$ denote their schemas, respectively.*

A	B	C	D
U_1	b	1	d_1
2	b	1	d_2

A	B	C	D
U_1	b	1	d_1
2	b	1	d_3

Considering the policies of Example 73, it is easy to check that $\cap^{S_1,S_2}(\rho^{S_2}(\rho^{S_1}(I)))$ returns the tuple $(2, b, 1, d_1)$. On the other hand, $\rho^{S'}(\cap^{S_1,S_2}(I))$, where S' denotes the schema of $R_1 \cap R_2$, returns the tuple $(U_1, b, 1, d_1)$ since this is the only tuple which is in both R_1 and R_2, and the policy has no effect. Hence, the two results are different; note also that neither of them is included in the other.

6.6 Experimental Results

We now describe several experiments we carried out to assess the effectiveness and the scalability of the index structures of Section 6.4. We compare our approach with a

Policy	Description
$\rho^{agg}(AVG, MAX, \mathit{AirTime}, \{\mathit{Origin}, \mathit{Dest}, \mathit{Carrier}\})$	Replace a missing flight air time with the average air time of the flights operated by the same carrier having the same origin and destination.
$\rho^{att}(AVG, MAX, MIN, \mathit{AirTime}, \mathit{ElapsedTime}, \{\mathit{Origin}, \mathit{Dest}, \mathit{Carrier}\})$	Replace a missing air flight time with the the air flight time corresponding to the average elapsed time of the flights operated by the same carrier having the same origin and destination.
$\rho^{reg}(MAX, AvgFare, \{City\}, \{Year, Quarter\})$	Determine a missing average fare (for a certain city in a certain quarter) by linear regression using the historical data for the same city.

Figure 6.4: Some of the PIPs used in the experiments

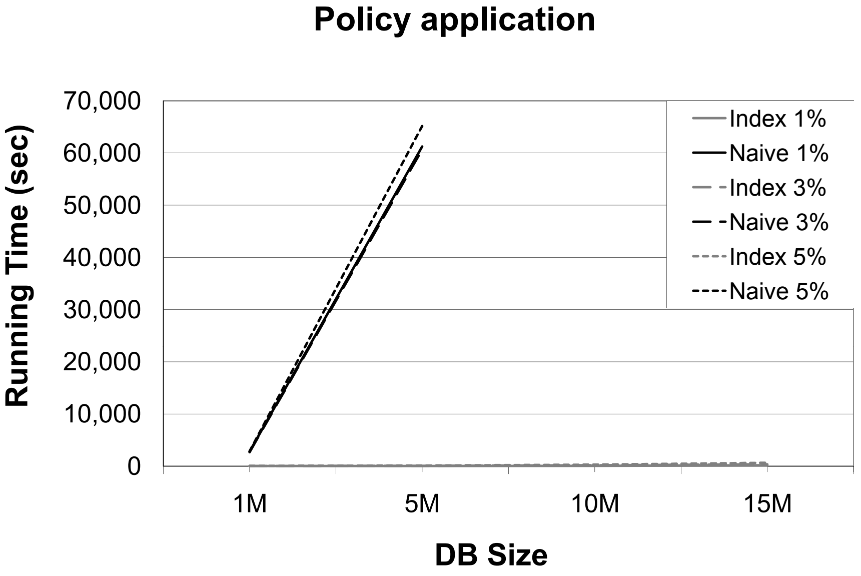


Figure 6.5: Policy application running time (different degrees of incompleteness)

naive one, the latter being a slight variant of Algorithm CT-ApplyPIP *not relying on the proposed indexes*. In order to make the application of a policy ρ faster with the naive approach, we defined a B-tree index on $sel(\rho)$ (we performed experiments showing that this speeds up the naive approach). We also compare the two approaches when they are combined with relational algebra operators and experimentally study the effects of the propositions in Section 6.5. Finally, we carried out an experimental evaluation of the quality of query answers with and without PIPs.

All experiments were carried out on a PostGres (v. 7.4.16) DBMS containing 20 years of U.S. flight data. The database schema has 55 attributes including date, origin,

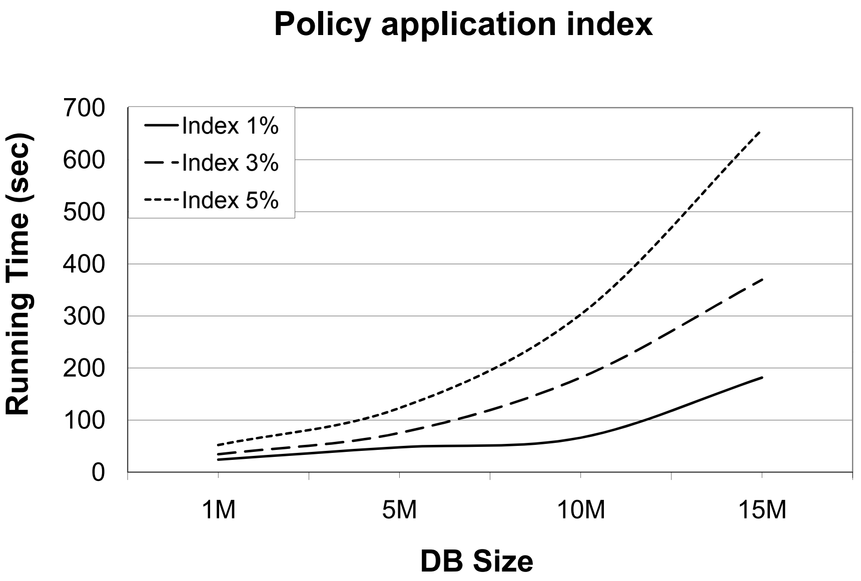


Figure 6.6: Policy application running time (different degrees of incompleteness)

destination, airborne time, elapsed time, carrier, etc. Experiments were run using multiple multi-core Intel Xeon E5345 processors at 2.33GHz, 8GB of memory, running the Scientific Linux distribution of the GNU/Linux OS kernel version 2.6.9-55.0.2.ELsmp. The index structures were implemented using Berkeley DB Java Edition. The algorithms for managing the index structures and applying policies in both approaches were written in JAVA. Some of the policies used in the experiments are reported in Figure 6.4. The results reported in this section apply only to aggregate policies; for the sake of brevity, we do not present the results for the other kinds of policies as they show the same trend.

6.6.1 Applying PIPs

We first compared the times taken by the two approaches to apply a policy. We varied the size of the DB up to 15 million tuples and the "amount of incompleteness" (percentage of rows with a null value) by randomly selecting tuples and inserting nulls (of different kinds) in them. For example, for an aggregate policy $\rho^{agg}(\mu, \nu, A, X)$ an $x\%$

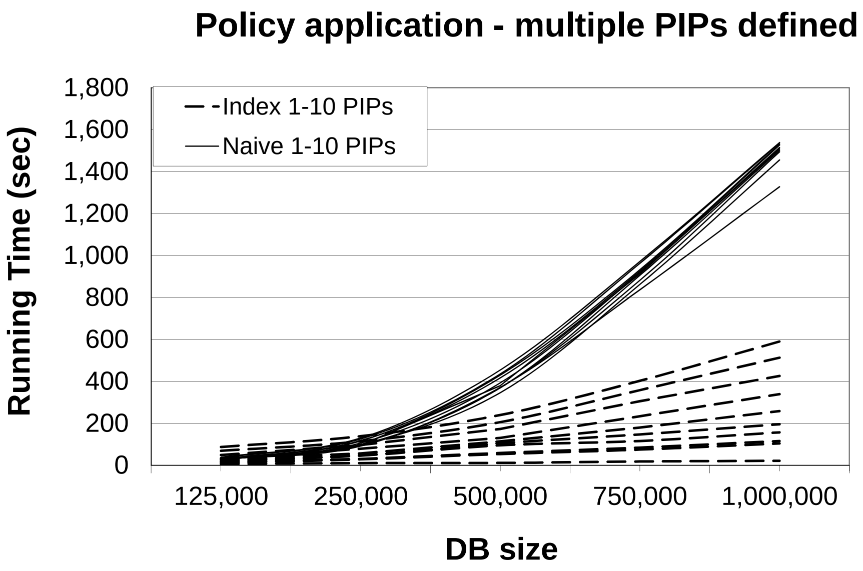


Figure 6.7: Running times of policy application with multiple policies defined

degree of incompleteness means that $x\%$ of the tuples in the database have null values in A.

Figure 6.5 shows the running times of policy application for different database sizes and three different amounts of incompleteness (only one policy is defined in this setting). It is important to note that the execution times for the index approach include both the time to apply a policy *and* the time taken to update the indexes. The gap between the two approaches increases significantly as the DB size increases with the index-based approach significantly outperforming the naive one – with 5 million tuples the former is 3 orders of magnitude faster than the latter. As expected, a higher degree of incompleteness leads to higher running times for both approaches. Figure 6.6 zooms in on the execution times for the index-based approach and shows that it scales well: able to manage databases up to 15 million tuples.

Figure 6.7 shows how execution times vary when multiple policies are defined (here the amount of incompleteness is 1%). The execution times of both approaches increase with the number of defined policies because additional data structures have to be updated when applying a policy. Our approach significantly outperforms the naive method.

These results show that our approach scales well when increasing DB size, amount of incompleteness and number of policies used – we can manage very large databases in a reasonable amount of time.

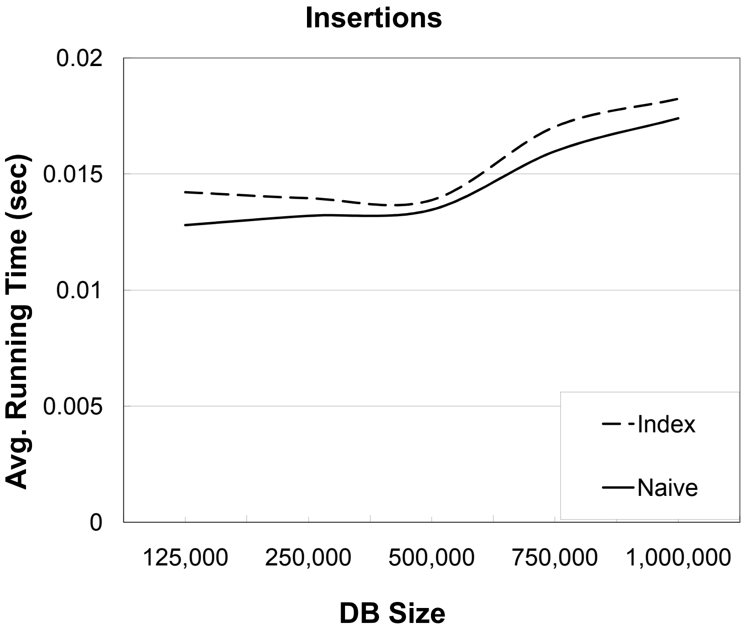


Figure 6.8: Tuple insertion running time

6.6.2 Updating the database

We also measured the time to execute tuple insertions, deletions, and updates; the results are shown in Figures 6.8–6.10. Each execution time is the average over at least 50 runs covering the different kinds of tuples that might be inserted, deleted or updated. The index-based approach is faster than the naive approach when tuple deletions are performed, but slower for tuple insertions and updates, though the differences are negligible and do not significantly increase as the database size increases. This small overhead is due to the management of the different data structures the index-based approach relies on

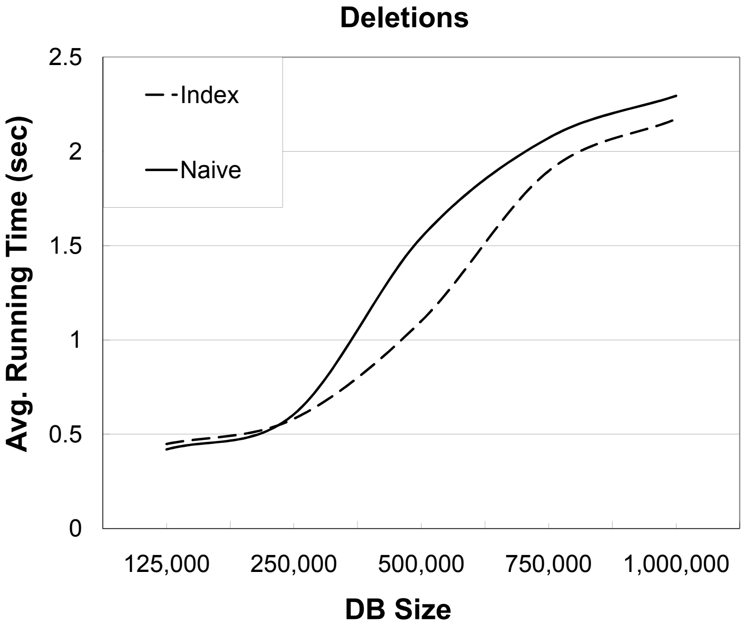


Figure 6.9: Tuple deletion running time

and is paid back by the better performances achieved for policy application, as discussed earlier. We further analyze this tradeoff in the following subsection.

6.6.3 Execution times under different loads

The results reported in the previous two sections show that policy applications are significantly faster with our index structures, but tuple insertions and updates are slightly slower (though tuple deletions are faster). Thus, the price we pay to maintain the indexes is when tuples are inserted and updated. Clearly, this cost gets higher as the number of modifications performed on the database increases, but it is paid back when policies are applied. We performed experiments with different loads of database modifications and policy applications to assess when the cost paid to perform the former is paid back by the time saved when the latter are executed. Specifically, we varied the number of modifications from 1000 to 10000 combining them with different numbers of policy applications. The experimental results are shown in Figure 6.11 (we used a database with 1 million

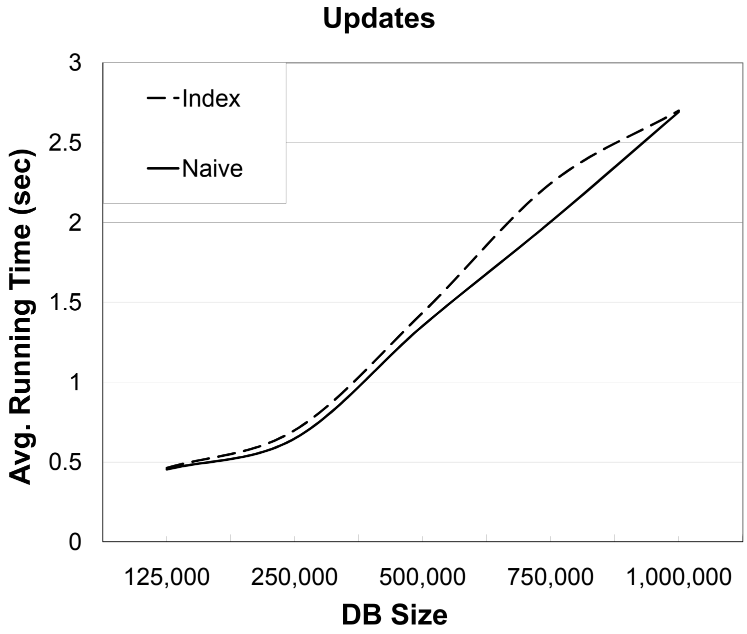


Figure 6.10: Tuple update running time

tuples and a 10% degree of incompleteness). The y-axis reports the difference between the running times of the naive and the index-based approaches. If only one (resp. two) policy application is performed, then the index running time gets higher than the naive one when more than 5000 (resp. around 10000) database modifications are applied. With more than two policy applications the index approach is always faster than the naive one up to 10000 modifications and, as shown by the trends of the curves, different thousands of modifications would be necessary to have the index approach slower than the naive.

6.6.4 Query answer quality

To assess the quality of query answers with and without policies, we performed an experimental evaluation using the World Bank/UNESCO database mentioned in the introduction. Specifically, we asked 5 analysts of our department (non computer scientist) who are working with this database and know it well to express 10 queries of interest over such data. As an example of query, they asked for the years during which the % of female

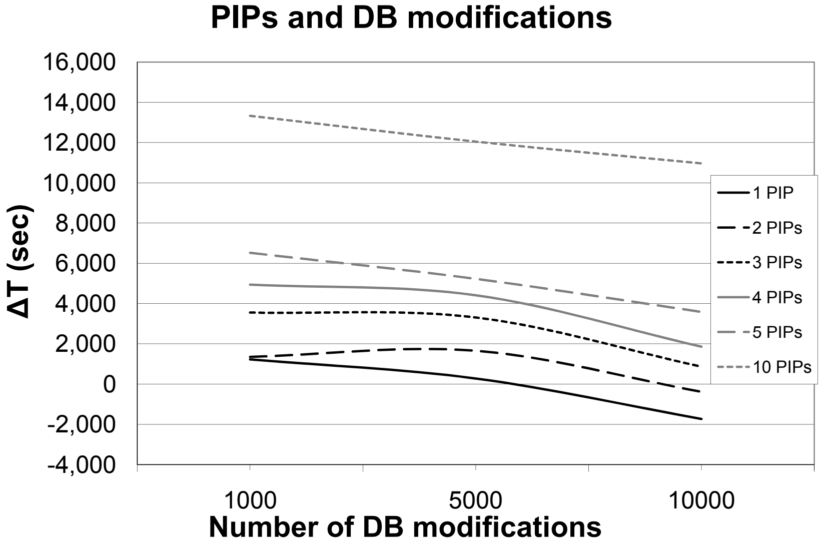


Figure 6.11: Execution times with different loads

unemployment was above a certain threshold and wanted to know what were the gross and under-age enrollment ratios in those years (to try to see if the gross and under-age enrollment ratios are somehow related to and affect the % of female unemployment). Furthermore, we asked the analysts to express different policies that would have been reasonable over such data and that captured some of their knowledge of the domain. Then, we asked them to rate the quality of query answers when policies are used and when the queries are evaluated on the original database without applying any policy. Specifically, users gave scores as integer numbers between 0 and 10, depending on their subjective evaluation of the quality of the results. The average score for each query is reported in Figure 6.12 and shows that end-users experience a substantial benefit when they can express how missing values should be replaced according to their needs and knowledge of the data. The higher quality of query answers when PIPs are used is generally due to

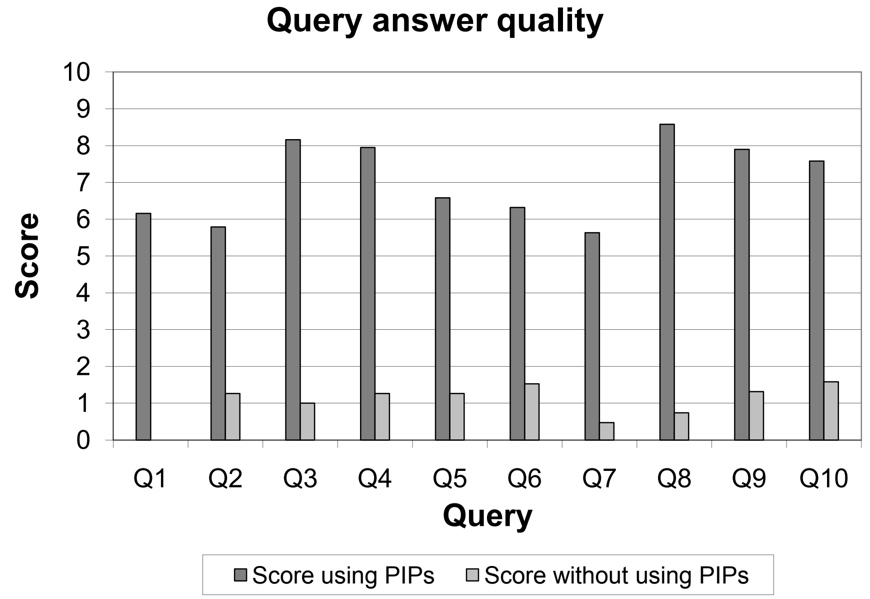


Figure 6.12: Query answer quality

the fact that more informative query answers are obtained after applying policies, that is, "more complete" tuples where null values have been filled according to the assumptions made by the user (and expressed in the policy) are returned to the user.

6.6.5 Relational Algebra operators and PIPs

We now compare the two approaches when they are combined with relational algebra operators. It is worth noting that applying a policy to the result of a relational algebra operator requires building index structures for the result database. We also wanted to experimentally see if there are substantial differences in execution times when a policy is applied before or after a relational algebra operator, under conditions which guarantee the same result (see Section 6.5), since this might be exploited for query optimization purposes.

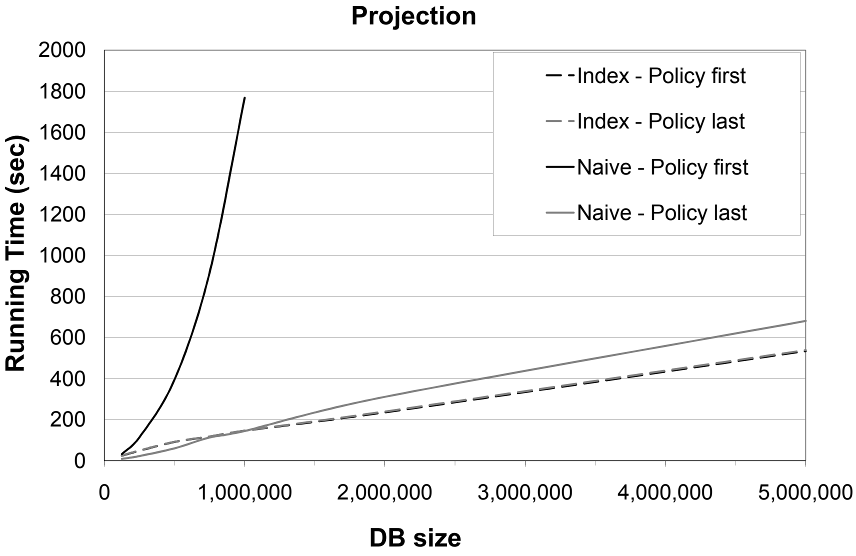


Figure 6.13: Running times of projection

We report on experiments for projection, join, and union as these are the three basic operators for which equivalence theorems exist. All experiments in this section were carried out on databases with 1% degree of incompleteness[5].

Projection. Figure 6.13 shows that applying the policy before or after projection makes little difference in running times when the indexes are adopted, whereas applying a policy after projection is more efficient when the naive approach is used. The index approach is slightly faster than the naive approach applied after projection, whereas it is much faster than the naive approach applied before projection.

Join. Figure 6.14 shows that applying a PIP after a join is more expensive than the other way around because the policy is applied to a much bigger relation. This difference is more evident for the naive approach. The fastest solution is applying a policy before join using the index structures.

[5] We performed the same experiments with 3% and 5% degrees of incompleteness and got the same trends as the ones reported here with just higher execution times due to the higher amount of incompleteness.

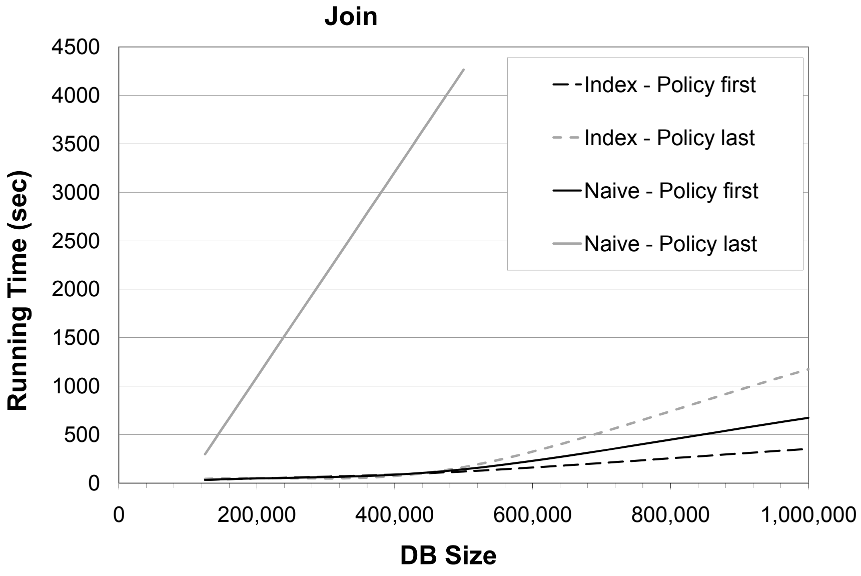


Figure 6.14: Running times of join

Union. Figure 6.15 shows that the index-based approach is faster than the naive approach regardless of the order in which policy and union are applied. Applying a policy before union gives better performance for the naive approach, whereas there is no significant difference for the index based approach.

To sum up, the index-based approach is faster than the naive one for all the relational algebra operators considered above. The gap between the two approaches gets bigger as the database size increases and thus, as the trends of the execution time curves show, it is expected to get even bigger with larger datasets.

6.7 Concluding Remarks

In all the works dealing with the management of incomplete databases, the DBMS dictates how incomplete information should be handled. End-users have no say in the matter. However, the stock analyst knows stocks, the market, and his own management's or client's attitude toward risk better than a DB developer who has never seen the stock

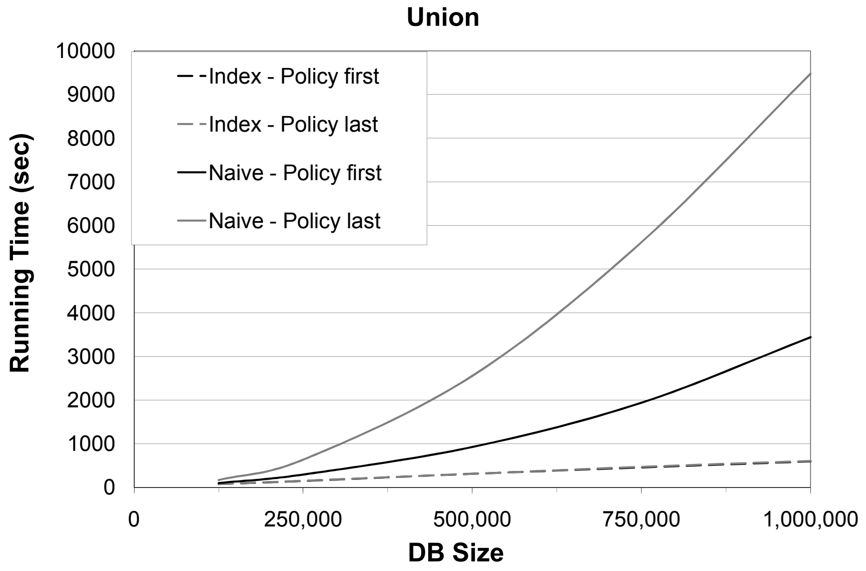


Figure 6.15: Running times of union

DB. He should make decisions on what to do with partial information, not the person who built the DBMS without knowing what applications would be deployed on it.

In this chapter, we propose the concept of a *partial information policy* (PIP). Using PIPs, end-users can specify the policy they want to use to handle partial information. We have presented examples of three families of PIPs that end-users can apply. We have also presented index structures for efficiently applying PIPs and conducted an experimental study showing that the adoption of such index structures allows us to efficiently manage very large datasets. Moreover, we have shown that PIPs can be combined with relational algebra operators, giving even more capabilities to users on how to manage their incomplete data.

Chapter 7

Query Answering under Uncertain Schema Mappings

The work described in this chapter appears in [GMSS09].

7.1 Introduction and Motivating Example

This chapter focuses on the problem of aggregate query processing across multiple databases in the presence of probabilistic schema mappings. The system may contain a number of data sources and a mediated schema, as in [DHY07]. Alternatively, a peer database system, with multiple data sources (*e.g.*, DB-life like information) and no mediated schema, as in [AKK$^+$03, HIM$^+$04] may also be in place.

There are many cases where a precise schema mapping may not be available. For instance, a comparison search "bot" that tracks comparative prices from different web sites has - in real time - to determine which fields at a particular location correspond to which fields in a database at another URL. Likewise, as in the case of [DHY07], in many cases, users querying two databases belonging to different organizations may not know what is the right schema mapping. We model this uncertainty about which schema

ID	price	agentPhone	postedDate	reducedDate
1	100k	215	1/5/2008	1/30/2008
2	150k	342	1/30/2008	2/15/2008
3	200k	215	1/1/2008	1/10/2008
4	100k	337	1/2/2008	2/1/2008

Table 7.1: An instance D_{S1}

mapping is correct by using probability theory. This robust model allows us to provide, in the case of aggregate queries, not only a ranking of the results, but also the expected value of the aggregate query outcome and the distribution of possible aggregate values.

We focus on five types of aggregate queries: COUNT, MIN, MAX, SUM, and AVG. Given a mediated schema, a query Q, and a data source S, Q is reformulated according to the (probabilistic) schema mapping between S's schema and the mediated schema, and posed to S, retrieving the answers according to the appropriate semantics (to be discussed shortly).

We focus on efficient processing of aggregate queries. An orthogonal challenge in this setting involves record linkage and cleansing that relates to duplicates. We assume the presence of effective tools for solving this problem [GIKS03, IKBS08] and focus on correct and efficient processing of the data. Also, we focus on the analysis of aggregate queries over a single table, to avoid mixing issues with joins over uncertain schema mappings. Our analysis tests the effect of executing an aggregate query over a single table or a table that is the result of any SPJ query over the non probabilistic part of the schema.

We define schema mappings between a source schema S and a target T in terms of attribute correspondences of the form $c_{ij} = (s_i, t_j)$, where s_i in S is the *source attribute* and t_i in T is the *target attribute*. For illustration purposes, we shall use the following two examples throughout the chapter:

Example 75. *Consider a real-estate data source S1, which describes properties for sale, their list price, an agent's contact phone, and the posting date. If the price of a property was reduced, then the date on which the most recent reduction occurred is also posted. The mediated schema T1 describes property list price, contact phone number, date of posting, and comments:*

S1 = (ID, price, agentPhone, postedDate, reducedDate)

T1 = (propertyID, listPrice, phone, date, comments)

*For the sake of simplicity, we assume that the mapping of ID to propertyID, price to listedPrice, and agentPhone to phone is known. In addition, there is no mapping to comments. Due to lack of background information, it is not clear whether date should be mapped to postedDate (denoted as mapping m_{11}) or reducedDate (denoted mapping m_{12}). Because of the uncertainty regarding which mapping is correct, we consider both mappings when answering queries. We can assign a probability to each such mapping (*e.g.*, m_{11} has probability 0.6 and m_{12} has probability 0.4). Such a probability may be computed automatically by algorithms to identify the correct mapping [CSD+08]. Table 7.1 shows an instance of a table D_{S1} of data source S1.*

Suppose that on February 20, 2008 the system receives a query Q1, composed on schema T1, asking for the number of "old" properties, those listed for more than a month:

```
Q1: SELECT COUNT(*) FROM T1
    WHERE date < '2008-1-20'
```

Using mapping m_{11}, we can reformulate Q1 into the following query:

```
Q11: SELECT COUNT(*) FROM S1
     WHERE postedDate < '2008-1-20'
```

transactionID	auctionID	time	bid	currentPrice
3401	34	0.43	195	195
3402	34	2.75	200	197.5
3403	34	2.8	331.94	202.5
3404	34	2.85	349.99	336.94
3801	38	1.16	330.01	300
3802	38	2.67	429.95	335.01
3803	38	2.68	439.95	336.30
3804	38	2.82	340.5	438.05

Table 7.2: An instance D_{S2}

Example 76. *As another example, consider eBay auctions. These auctions have a strict end date for each auction and use a second-price model. That is, the winner is the one who places the highest bid, but the winning price is (a delta higher than) the second-highest bid. Now consider two (simplified) database schemas, S2 and T2, that keep track of auction prices:*

S2 = (transactionID, auction, time, bid, currentPrice)

T2 = (transaction, auctionId, timeUpdate, price)

For simplicity, we again assume that the mappings of transactionID to transaction, auction to auctionID and the mapping of time to timeUpdate are known. The attribute price in T2 can be mapped to either the bid attribute (denoted as mapping m_{21}) or the currentPrice attribute (denoted as mapping m_{22}) in S2. Here, the source of uncertainty may be attributed to the sometimes confusing semantics of the bid and the current price in eBay auctions. Assume that m_{21} is assigned probability 0.3 and m_{22} is assigned probability 0.7. Table 7.2 contains data for two auctions (numbers 34 and 38) with four bids each. The time is measured from the beginning of the auction and therefore 0.43 means that about 10 hours (less than half a day) have passed from the opening of the auction. Sup-

pose that the system receives a query Q2 w.r.t. schema T2, asking for the average closing price of all auctions:

```
Q2: SELECT AVG(R1.price)  FROM
(SELECT MAX(DISTINCT R2.price)
   FROM T2 AS R2
   GROUP BY R2.auctionID) AS R1
```

The subquery, within the FROM clause, identifies the maximum price for each auction. Using mapping m_{21}, we can reformulate Q2 to be:

```
Q21: SELECT AVG(R1.currentPrice) FROM
(SELECT MAX(DISTINCT R2.currentPrice)
   FROM T2 AS R2
   GROUP BY R2.auction) AS R1
```

As mentioned in Section 2.4, two different semantics have been proposed for dealing with query answering using probabilistic schema matchings [DHY07, DSDH08]: a "by-table" semantics and a "by-tuple" semantics. We analyze aggregates `COUNT`, `MIN`, `MAX`, `SUM`, and `AVG` and define three semantics for such aggregate functions that we combine with by-table and by-tuple semantics. In the first one, an aggregate query returns a set of possible values for the answer, together with a probability distribution over that set. We call this the "distribution" semantics. A second method returns just a *range* specifying the lowest and highest possible values for the aggregate query. We call this the "range" semantics. The third semantics returns an *expected value*. In this work, we first propose these three semantics for aggregate computations and then show that they combine with the by-table and by-tuple semantics of [DHY07] in six possible ways, yielding six possible semantics for aggregates in probabilistic schema mapping. We develop algorithms to compute under each of the six semantics and show that the algorithms are correct. We develop a characterization of the computational complexity of the problem of computing these six semantics. For all the above aggregate operators, we show that semantics based on the by-table semantics are PTIME computable. For the COUNT operator, we show that query results for all six semantics can be computed in PTIME. Computing the SUM operator is in PTIME for all but the by-tuple/distribution semantics. Finally, we show that for MIN, MAX, and AVG, the only by-tuple semantics that can be efficiently computed is the range semantics.

We have developed a prototype implementation of our algorithms and tested out their efficiency on large data sets, showing that our algorithms work very efficiently in practice. Our experiments show the computational feasibility of the different semantics for each of the aggregate operators mentioned above. We show that, for each aggregate operator considered in this work under the by-tuple semantics, the algorithms for computing the range semantics are very efficient and scalable; this is also the case for COUNT under the other two semantics. Furthermore, the expected value semantics for SUM is also very efficient since we can take advantage of the fact that it is guaranteed to be equivalent to the by-table semantics, as we show in this work. In summary, for each aggregate operator, there is at least one semantics where our experiments show that it can be computed very efficiently.

To summarize, our contributions are as follows:

1. We show six possible semantics for aggregate queries with uncertain schema mappings.
2. We show several cases under the by-tuple semantics where efficient algorithms exist for aggregate computation.

3. We prove that for the SUM aggregate operator, by-tuple/expected value and by-table/expected value semantics yield the same answer.

4. Using a thorough empirical setup, we show that the polynomial time algorithms are scalable up to several million tuples (with some even beyond 30 million tuples) and with a large number of mappings.

The rest of the chapter is organized as follows. Section 7.2 provides background on aggregate query answering under uncertain schema mapping. The six semantics for aggregate query processing in the presence of uncertain schema mappings is described in detail in Section 7.3. Section 7.4 provides a set of algorithms for efficient computation of the various aggregates. Our empirical analysis is provided in Section 7.5. We conclude directions for future work presented in Section 7.6 and final remarks in Section 7.7.

7.2 Preliminaries

We base our model of probabilistic schema mappings on the one presented in [DHY07], extending it to answer aggregate queries. In what follows, given relational schemas $\overline{S}$ and $\overline{T}$, S a relation in $\overline{S}$, and T a relation in $\overline{T}$, an attribute correspondence is a one-to-one mapping from the attribute names in S to the attribute names in T. Also, a one-to-one relation mapping is a mapping where each source and target attribute occurs in at most one correspondence.

Definition 65 (Schema Mapping)**.** *Let $\overline{S}$ and $\overline{T}$ be relational schemas. A relation mapping M is a triple (S, T, m), where S is a relation in $\overline{S}$, T is a relation in $\overline{T}$, and m is a set of attribute correspondences between S and T.*

A schema mapping *$\overline{M}$ is a set of one-to-one relation mappings between relations in $\overline{S}$ and in $\overline{T}$, where every relation in either $\overline{S}$ or $\overline{T}$ appears at most once.*

The following definition, also from [DHY07], extends the concept of schema mapping with probabilities:

Definition 66 (Probabilistic Mapping)**.** *Let* $\overline{S}$ *and* $\overline{T}$ *be relational schemas. A* probabilistic mapping (p-mapping) pM *is a triple* $(S, T, \mathbf{m})$*, where* $S \in \overline{S}$*,* $T \in \overline{T}$*, and* $\mathbf{m}$ *is a set* $\{(m_1, Pr(m_1)), ..., (m_l, Pr(m_l))\}$*, such that*

- *for* $i \in [1, l]$*,* m_i *is a one-to-one relation mapping between* S *and* T*, and for every* $i, j \in [1, l]$*,* $i \neq j \Rightarrow m_i \neq m_j$*.*
- $Pr(m_i) \in [0, 1]$ *and* $\sum_{i=1}^{l} Pr(m_i) = 1$*.*

A schema p-mapping $\overline{pM}$ *is a set of p-mappings between relations in* $\overline{S}$ *and in* $\overline{T}$*, where every relation in either* $\overline{S}$ *or* $\overline{T}$ *appears in at most one p-mapping.*

7.3 Semantics

We now present the semantics of aggregate queries in the presence of probabilistic schema mappings. We start with a formal presentation of the by-table and by-tuple semantics, as introduced in [DHY07] (Section 7.3.1). Then, we move on to introduce three aggregate semantics and their combination with the by-table and by-tuple semantics (Section 7.3.1).

7.3.1 Semantics of Probabilistic Mappings

The intuitive interpretation of a probabilistic schema mapping is that there is uncertainty about which of the mappings is the *right one*. Such uncertainty may be rooted in the fact that "the syntactic representation of schemas and data do not completely convey the semantics of different databases," [MHH00] *i.e.*, the description of a concept in a schema can be semantically misleading. As proposed in [DHY07], there are two ways in which this uncertainty can be interpreted: either a single mapping should be applied to the entire set of tuples in the source relation, or a choice of a mapping should be made for each of these tuples. The former is referred to as the *by-table* semantics, and the latter as the *by-tuple* semantics. The *by-tuple* semantics represents a situation in which data is gathered from multiple sources, each with a potentially different interpretation of a schema.

As discussed in [DHY07], the high complexity of query answering under the by-tuple semantics is due to the fact that all possible *sequences* of mappings (of length equal to the number of tuples in the table) must be considered in the general case. The following examples illustrate the difference between the two semantics when considering aggregate functions.

Example 77. *Consider the scenario presented in Example 75. Assume the content of table* D_{S1} *is as shown in Table 7.1. Using the two possible mappings, we can reformulate* Q1 *into the following two queries, one for each possible way of mapping attribute* date*:*

```
Q11: SELECT COUNT(*) FROM S1
          WHERE postedDate < '2008-1-20' Q12: SELECT COUNT(*)
FROM S1
          WHERE reducedDate < '2008-1-20'
```

We can adapt the procedure described for the by-table semantics in [DHY07] to answer uncertain aggregate queries, by computing each of the two previous reformulated queries as if they were the correct mappings and the probability of the corresponding mapping is assigned to each answer. In this case, the system provides answer 3 *with probability* 0.6 *(from query* Q11*) and answer* 2 *with probability* 0.4 *(from query* Q12*). Under the by-tuple semantics it is necessary to consider all possible sequences,* i.e.*, ways of assigning* *a mapping to a tuple. For instance, the sequence $s = \langle m_{11}, m_{12}, m_{12}, m_{11}\rangle$ represents the fact that tuple 1 and 4 should be interpreted under mapping m_{11}, in which case attribute* date *is mapped to* postedDate*, and tuples 2 and 3 should be interpreted using mapping m_{12} which maps* date *to* reducedDate*. Each sequence has an associated probability equal to the product of the probability of each mapping in the sequence, since mappings are independently assigned to tuples. For instance, the probability of sequence s is*

$$Pr(s) = 0.6 * 0.4 * 0.4 * 0.6 = 0.0576$$

An answer in this case, as discussed in [DHY07] for general SPJ queries, can be obtained by computing the aggregate operator for each possible sequence. The final answer is a table that contains all the different values obtained from the answers yielded by each individual computation, each with an associated probability. The probability for each value is the sum of the probabilities of all sequences that yield that value. In this example, the final answer is 1 with probability 0.16, 2 with probability 0.48, and 3 with probability 0.36.

Example 78. *Let us now consider Table 7.2 and query* Q2*, presented in Example 76. Using the two possible mappings, we can reformulate* Q2 *into the following two queries:*

```
Q21: SELECT AVG(R1.currentPrice) FROM
(SELECT MAX(DISTINCT R2.currentPrice)
   FROM T2 AS R2
   GROUP BY R2.auction) AS R1
Q22: SELECT AVG(R1.bid) FROM
(SELECT MAX(DISTINCT R2.bid)
FROM T2 AS R2
GROUP BY R2.auction) AS R1
```

Using the by-table semantics, the system provides the answer 345.245 *with probability* 0.3 *and* 385.945 *with probability* 0.7. *Under the by-tuple semantics, in order to compute an answer to* Q2*, given that there are 8 tuples in the database instance and 2 possible mappings, we have to look at* $2^8 = 256$ *sequences. We need to compute the answer for each sequence and then combine the results.*

Semantics for Aggregate Queries Under Uncertain Schema Mappings

Aggregate queries provide users with answers that are not simple cut & paste data from the database. Rather, data is processed and user expectations are also different. In many cases, users expect a simple, single answer to an aggregate query (*e.g.*, counting the number of newly posted houses). Therefore, when extended to probabilistic schema mappings, such expectations should be taken into account.

In this work, we consider three common extensions to semantics with aggregates and probabilistic information. The *range semantics* gives an interval within which the aggregate is guaranteed to lie. The *distribution semantics* specifies all possible values that the aggregate can take, and for each such value, it gives the probability that it is the correct one. Of course, we can easily derive the answer to an aggregate query under the range semantics from the answer to the same query under the distribution semantics. Finally, for those who like the answer to be a single number, we develop an *expected value* semantics which returns the expected value of the aggregate. Note that the answer to a query under the expected value semantics can also be computed from the answer to the query under the distribution semantics. In a sense, the answer according to the distribution semantics is rich, containing details that are eliminated in the other two semantics. However, as we will see below, the other two semantics may be more efficiently computable without obtaining the distribution at all.

Let $\mathbf{m} = \{(m_1, Pr(m_1)), ..., (m_l, Pr(m_m))\}$ be a set of all possible mappings from schema S to schema T, each with an associated probability $Pr(m_i)$, where $\sum_i Pr(m_i) = 1$. Let $V = \{v_1, ..., v_n\}$ be the set of results of evaluating the aggregate function for each possible mapping or a sequence of mappings. The three possible semantics for query answering with aggregate functions and multiple possible schema mappings can be formalized as follows:

1. *Range Semantics:* The result of the aggregate function under the *range semantics* is the interval $[\min(V), \max(V)]$.

2. *Probability Distribution Semantics:* Under the *probability distribution* semantics, the result of the aggregate function is a random variable X. For every distinct value $r_j \in V$, we have that

$$Pr(X = r_j) = \sum_{v_i \in V, v_i = r_j} Pr(m_i) \tag{7.1}$$

3. *Expected Value Semantics:* Let $V = \{v_1, ..., v_n\}$ be the set of results of evaluating the aggregate function for each possible mapping. The result of the aggregate function under the *expected value semantics* is

$$\sum_{i=1}^{n} Pr(m_i) * v_i \tag{7.2}$$

The fact that answers to queries under the range and expected value semantics can be immediately derived from the answer under the distribution semantics tells us

```
Algorithm ByTableAggregateQuery
Input: Table $S, T$; MapList $M$; Attribute $A$; Condition $C$;
       AggregateFunction $Agg$; Semantics $S$;

Let $|M| = l$ be the number of mappings for attribute $A$;
Let $A_1, ..., A_l$ be all the attributes to which $A$ maps;
For $i = 0$ to $l$,
  Let $r_i$ be the answer for the query:
    SELECT Agg($A_i$) FROM T WHERE C GROUP BY B;
return CombineResults($r_1, ..., r_l, S$);
```

Figure 7.1: Generic by-table algorithm adapted from Halevy's work for Aggregate Queries

that if the distribution semantics is PTIME computable, then the range and expected value semantics should also be PTIME computable.

Possible Combinations of Semantics. When combining the by-table and by-tuple semantics with the three aggregate semantics suggested in Section 7.3.1, a space of six possible semantics for aggregate queries over probabilistic schema mappings is created. This space is illustrated in Table 7.3, where for each semantics we give the query answer to query Q1.

COUNT	Range	Distribution	Exp. Value
By-Table	$[2, 3]$	3 (prob 0.6), 2 (prob 0.4)	2.6
By-Tuple	$[1, 3]$	see Example 77	2.2

Table 7.3: The Six Semantics of Aggregate Queries over Probabilistic Schema Mapping

7.4 Algorithms for Aggregate Query Answering

7.4.1 By-Table Semantics

Figure 7.1 provides a "generic" algorithm to answer aggregate queries under the by-table semantics, extending a similar algorithm in [DHY07]. The algorithm reformulates

```
Algorithm ByTupleRangeCOUNT
Input: Table $S, T$; MapList $M$; Attribute $A$; Condition $C$;
       AggregateFunction $Agg$; Semantics $S$;

Let $up$ an $low$ be equal to 0;
For each $t_i \in S$,
  if for all mappings $m_j \in M$ such that $t_i$ satisfies $C$ then
    $low = low + 1; up = up + 1;$
  else if there exists a mapping $m_j \in M$ for which $t_i$ satisfies $C$ then
    $up = up + 1;$
return $[low, up]$;
```

Figure 7.2: Algorithm to answer `SELECT COUNT(A) FROM T WHERE C` under Range Semantics

the input query into l new queries, one for each possible schema mapping and obtains an answer r_i to the query w.r.t. that mapping. Finally, it outputs the result using function `CombineResults`. `CombineResults` returns $[min(r_1, \ldots, r_m), max(r_1, \ldots, r_m)]$ when the semantics chosen is the range semantics. When the semantics chosen is the expected value semantics, it returns $\Sigma_{i=1}^{m} Pr(m_i) * r_i$ where $Pr(m_i)$ is the probability that the mapping that maps A to A_i is correct. When the semantics chosen is the distribution semantics, it returns the set of all pairs $\{(r_i, p) \mid p = \Sigma_{r_j = r_i} Pr(m_i)\}$.

7.4.2 By-Tuple Semantics

The by-tuple semantics associates a mapping with each tuple in a relational table. Hence, if we have n tuples and m different mappings, there are m^n different sequences that assign mappings to tuples. The problem of answering select, project, join queries under the by-tuple semantics is in general #P-complete in data complexity [DHY07]. The reason for the high complexity stems from the need to assign probabilities to each tuple. Computing all by-tuple answers without returning the probabilities is in PTIME. When it comes to aggregate queries, however, merely computing all possible tuples is not enough. One also needs to know, for each possible mapping sequence, whether a tuple belongs to it or not. Therefore, in the worst case, going through all possible mapping sequences is unavoidable. To see why, consider the following query against Table 7.2:

```
SELECT SUM(price) FROM T2
```

With 2 possible mappings and 8 tuples, there are $2^8 = 256$ possible sequences. In this case, there are 128 different possible values — in fact, there would have been 256 different possible values if the bid and currentPrice of the first tuple did not have the same value (195). Therefore, merely enumerating all possible answers may yield an exponential number of answers.

The generic (naïve) algorithm discussed earlier can be greatly improved when we consider specific aggregate functions. In this section, we show how to achieve this for the COUNT, SUM, AVG, MAX, and MIN aggregate functions under the three alternative semantics presented in Section 7.3. We show that in certain aggregate/semantics combinations, it is possible to compute an answer in PTIME, whereas for others PTIME algorithms could not be found.

Aggregate function COUNT. We present algorithms to compute the COUNT aggregate under by-tuple/range and by-tuple/distribution semantics. The answer for the expected value semantics can be computed directly from the result provided by the algorithm for distribution semantics.

We will use our running examples presented in Section 7.2. Consider the setting from Example 75, the data in Table 7.1, and query Q1:

```
SELECT COUNT(*) FROM T1
          WHERE date < '1-20-2008'
```

COUNT *Under the Range Semantics.* Under the range semantics, the answer to query Q1 should provide the minimum and the maximum value for the aggregate, considering any

```
**Algorithm** *ByTuplePDCOUNT*
**Input:** Table $S, T$; MapList $M$; Attribute $A$; Condition $C$;

Let $pd$ be a new probability distribution;
In $pd$ set $Pr(0) = 1.0$;
**For each** $t_i \in S$,
  Let *occProb* be the sum of the probabilities of mappings in $M$
    under which $t_i$ satisfies $C$;
  Let *notOccProb* be the sum of the probabilities of mappings in $M$
    under which $t_i$ does not satisfy $C$;
  In $pd$ set $Pr(0) = Pr(0) *$ *notOccProb*;
  **For** $j = 1$ to $i - 1$,
    In $pd$ set $Pr(j) = (Pr(j) *$ *notOccProb*$) + (Pr(j-1) *$ *occProb*$)$;
  In $pd$ set $Pr(i) = Pr(i-1) *$ *occProb*;
**return** $pd$;
```

Figure 7.3: Algorithm to answer `SELECT COUNT(A) FROM T WHERE C` under Distribution Semantics

tupleID	*low*	*up*	*comment*
	0	0	initialization
1	0	1	cond. satisfied under m_{11}
2	1	2	cond. satisfied under both mappings
3	1	2	cond. satisfied under no mapping
4	1	3	cond. satisfied under m_{11}

Table 7.4: Trace of `ByTupleRANGE` for query Q1

of the mappings. The algorithm is shown in Figure 7.2. The idea behind the algorithm is simple: each tuple, depending on the mapping that is used for it, may or may not satisfy the selection condition for the `COUNT`. Clearly, if a tuple satisfies the select condition under all mappings, then both the minimum and maximum possible values for `COUNT` should be increased. If the tuple does not satisfy the select condition under all mappings, then it is never included in the aggregate result. Finally, if there is at least one mapping under which the tuple does not satisfy the select condition, then the minimum value does not change, but the maximum does.

To see how this algorithm works, we include in Table 7.4 the trace of how the bounds are updated with each tuple in Table 7.1 to answer query Q1. For instance, we can see that for tuple 1 only the upper bound is incremented because this tuple satisfies the select condition only for mapping m_{11}. The last row of the table shows the final answer, $[1, 3]$.

Note that this algorithm looks at each tuple only once, and in each step it looks at most at all mappings once. Thus, if n is the number of tuples in S and m is the number of possible mappings, the number of computations needed for this algorithm is in $O(n * m)$.

Theorem 21. *Algorithm* `ByTupleRangeCOUNT` *correctly computes the result of executing a* `COUNT` *query under the by-tuple range semantics.*

Proof. Suppose, towards a contradiction, that the algorithm returns the range $[\ell, u]$ and that there exists a possible answer k such that either $k < \ell$ or $u < k$. If k is in fact a possible value for the COUNT query, then this means that there are k tuples in T such that for each of these tuples there exists at least one mapping sequence for which the selection condition is valid. However, the algorithm increases the current value of the upper bound every time it finds a tuple for which the selection is valid under at least one mapping, meaning that these k tuples will be considered and thus $k \leq u$. For the lower bound, the fact that the algorithm returned ℓ means that it found ℓ tuples in T such that for each of them the selection condition was true under any mapping, which contradicts the hypothesis that $k < \ell$. □

`COUNT` *Under the Distribution Semantics.* A naïve way of computing an answer for a query such as Q1 under the distribution semantics is to consider all possible sequences of mappings and to compute the query for each sequence, as shown in the second part of Example 77. However, we present a more efficient algorithm that only takes polynomial time in the number of mappings and the number of tuples in the table. The pseudo-code of this algorithm is outlined in Figure 7.3.

tupleID	0	1	2	3	4
1	0.4	0.6			
2	0.4	0.6	0		
3	0	0.4	0.6	0	
4	0	0.16	0.48	0.36	0

Table 7.5: Trace of ByTuplePDCOUNT for query Q1

Under a given mapping, a tuple can either add 0 to the COUNT result or 1. Hence, the probability of a tuple adding 1 to the result is that of the mapping itself, and the probability of adding nothing is the complementary probability. This reasoning can be easily extended to multiple mappings by taking the sum of the probabilities for which the tuple adds 1 to the calculation. If we look at each tuple in turn, the value of the aggregate at a certain time depends on how many tuples were taken into account. However, at each step, the count can at most be incremented by one, depending on whether the tuple at hand satisfies the selection condition. This means that if we are looking at tuple i, and the count so far is c_{i-1}, then after looking at tuple i the count will either be c_{i-1} or $c_{i-1} + 1$. Since this can be the case at each update, we must store all possible values for the result at each step. For instance, after looking at just one tuple, only two values are possible (0 and 1), and when we look at another tuple, the value 2 now becomes possible. The probabilities associated with each of these results can be easily updated at each step by looking at two values as shown in the algorithm.

Table 7.5 shows the trace of how the probability distribution is updated with each tuple in Table 7.1 to answer query Q1. For instance, consider the second row in the table, where tuple 2 is processed. This tuple has probability 0 of being part of the result because under both mappings it does not satisfy the select condition. The probability of the result being 0 is now 0.4; this is because the count can only be 0 if it was 0 before and tuple 2 is not part of the count ($0.4 * 1.0 = 0.4$); the probability of the result being 1 is updated in the following way: the value can only be 1 if either it was 0 before and tuple 2 satisfies the condition, or it was already 1 and tuple 2 does not satisfy the condition ($0.4 * 0 + 0.6 * 1.0 = 0.6$). Finally, 2 is a new possible value with probability 0 for now. Note that each row is a probability distribution among the values considered thus far. The final probability distribution is the same as shown in Example 77.

Theorem 22. *Algorithm `ByTuplePDCOUNT` correctly computes the result of executing a `COUNT` query under the by-tuple/distribution semantics.*

Proof. We will prove this statement by induction on the number of tuples in T. For $|T| = 0$, the probability distribution given by the algorithm is trivially correct since the answer can only be 0.

Suppose now that the statement holds for all tables T such that $|T| = k, k \in N, k > 1$. We must now prove that the statement holds for $|T| = k + 1$. Let T' be equal to T without its last tuple; since $|T'| = k$, the algorithm correctly computes a probability distribution pd for the answer to the query.

Now, let `occProb` and `notOccProb` be the values calculated by the algorithm during its last iteration of the *for* loop in line 3, *i.e.* for the last tuple in T. Since pd is correct, for any $0 \leq i \leq k$ the value $pd(i)$ is equal to the sum of the probabilities of all mapping sequences under which the result is i, *i.e.* $pd(i) = Pr(s_1^i) + ... + Pr(s_y^i)$, where $s_1^i, ..., s_y^i$ are all the sequences that yield answer i. Now, after updating pd in line 8 of the algorithm, we get $pd(i) = (Pr(s_1^i) + ... + Pr(s_y^i)) * \textit{notOccProb} + (Pr(s_1^{i-1}) + ... + Pr(s_y^{i-1})) * \textit{occProb}$. If we distribute the multiplication signs with respect to the sums, we get $Pr(s_1^i) * \textit{notOccProb} + ... + Pr(s_y^i) * \textit{notOccProb} + Pr(s_1^{i-1}) * \textit{occProb} + ... + Pr(s_y^{i-1}) * \textit{occProb}$. Since *notOccProb* is the sum of the probabilities of the mappings under which the last tuple is not part of the count, each term $Pr(s_j^i) * \textit{notOccProb}$ represents the sum of the probabilities of the sequences starting with s^i_j under which the answer is i, and analogously for the terms multiplied by *occProb*. Since these are all the possible sequences that can yield a value of i, we conclude that the probability is computed correctly.

Since the argument above was built on an arbitrarily chosen value $0 \leq i \leq k$, we conclude that the probability distribution is correct for all such values. Finally, for $pd(k{+}1)$ the reasoning is analogous, except that the summation multiplied by *notOccProb* is zero because it can never yield a value of $k + 1$. □

In Section 7.3.1, the probability distribution for this example was computed by looking at the answer of each possible sequence of mappings assigned to individual tuples. If we have m mappings and n tuples, then the number of sequences is m^n. The algorithm presented here is polynomial in the number of mappings and tuples, and the number of computations is in $O(m * n^2)$.

Aggregate functions SUM and AVG

We now present efficient (PTIME) algorithms to compute the `SUM` aggregate under the by-tuple/range and by-tuple/expected value semantics. Computing this aggregate function under the distribution semantics does not scale, simply because the number of newly generated values may be exponential in the size of the original table, as was demonstrated at the beginning of Section 7.4.2.

`SUM` *Under the Range Semantics.* For the range semantics, we must compute the tightest interval in which the aggregate lies. The algorithm is presented in Figure 7.4 and illustrated next.

Consider Example 76, but now suppose we are interested in a simple computation of the sum of the prices for transactions whose auctionID is 34; we then use the following query:

```
Algorithm ByTupleRangeSUM
Input: Table $S, T$; MapList $M$; Attribute $A$; Condition $C$;

Let low $= 0$, up $= 0$;
For each $t_i \in S$,
    Let $v_i^{min}$ be the minimum value obtained by applying a mapping in $M$
        that satisfies condition $C$;
    Similarly, let $v_i^{max}$ be the maximum value that satisfies condition $C$;
    low $=$ low $+ v_i^{min}$;
    up $=$ up $+ v_i^{max}$;
return [low, up];
```

Figure 7.4: Algorithm to answer `SELECT SUM(A) FROM T WHERE C` under Range Semantics

tupleID	v_i^{min}	v_i^{max}	*low*	*up*
			0	0
1	195	195	195	195
2	197.5	200	392.5	395
3	336.3	439.95	728.8	834.95
4	340.5	438.05	1069.3	1273

Table 7.6: Trace of `ByTupleRANGE` for query Q2'

Q2': `SELECT SUM(`price`) FROM` T2

`WHERE` auctionID = `'34'`

Table 7.6 shows the trace of the algorithm in Figure 7.4 to answer query Q2'. If we look, for instance, at the second row in the table, processing tuple 2 from Table 7.2, $v_2^{min} = 197.5$ and $v_2^{max} = 200$, thus $low = 392.5$ and $up = 395$. The answer to Q2' is thus $[1069.3, 1273]$. This algorithm is polynomial in the number of mappings and tuples, and the number of computations is in $O(m * n)$, where m is the number of mappings and n is the number of tuples.

Theorem 23. *Algorithm `ByTupleRangeSUM` correctly computes the result of executing a `SUM` query under the by-tuple/range semantics.*

Proof. Suppose, towards a contradiction, that the algorithm returns the range $[\ell, u]$ and that there exists a possible answer k such that either $k < \ell$ or $u < k$. Since ℓ, u, and k are all sums of values from tuples in T, if k is in fact a possible value for the SUM query, this means that there is at least one tuple in T such that, for some mapping, the value of A is less (respectively, greater) than v_i^{min} (respectively, v_i^{max}). This is a contradiction since the algorithm chooses this value to be the minimum (respectively, maximum) under all possible mappings. □

`AVG` *under the Range Semantics.* For the `AVG` aggregate operator, the algorithm we developed is very similar to the one in Figure 7.4, keeping a counter of the number of participating tuples for both the lower bound and the upper bound. The counter for the upper bound is incremented by one at each step only if there exists a maximum value for the tuple that satisfies the condition when some mapping is applied. The counter for the lower bound is incremented only if there is a minimum value for the tuple that satisfies the condition under some mapping. The answer is given by dividing each bound for `SUM` by the corresponding counter.

Theorem 24. *Algorithm* `ByTupleRangeAVG` *correctly computes the result of executing an* `AVG` *query under the by-tuple/range semantics.*

Proof. Since the ByTupleRangeAVG algorithm is a trivial variation of the ByTupleRange-SUM algorithm to count the number of tuples satisfying the condition, this result is a direct consequence of Theorem 23. □

`SUM` *Under the Expected Value Semantics.* We now address an efficient way of computing by-tuple/expected value semantics. We do so not by giving an algorithm, but rather by showing that an answer to a `SUM` query using the by-tuple/expected value semantics is

Sequence	SUM	p	SUM$\times p$
$(m_{21}, m_{21}, m_{21}, m_{21})$	1076.93	0.0081	8.723133
$(m_{21}, m_{21}, m_{21}, m_{22})$	1063.88	0.0189	20.107332
$(m_{21}, m_{21}, m_{22}, m_{21})$	947.49	0.0189	17.907561
$(m_{21}, m_{21}, m_{22}, m_{22})$	934.44	0.0441	41.208804
$(m_{21}, m_{22}, m_{21}, m_{21})$	1074.43	0.0189	20.306727
$(m_{21}, m_{22}, m_{21}, m_{22})$	1061.38	0.0441	46.806858
$(m_{21}, m_{22}, m_{22}, m_{21})$	944.99	0.0441	41.674059
$(m_{21}, m_{22}, m_{22}, m_{22})$	931.94	0.1029	95.896626
$(m_{22}, m_{21}, m_{21}, m_{21})$	1076.93	0.0189	20.353977
$(m_{22}, m_{21}, m_{21}, m_{22})$	1063.88	0.0441	46.917108
$(m_{22}, m_{21}, m_{22}, m_{21})$	947.49	0.0441	41.784309
$(m_{22}, m_{21}, m_{22}, m_{22})$	934.44	0.1029	96.153876
$(m_{22}, m_{22}, m_{21}, m_{21})$	1074.43	0.0441	47.382363
$(m_{22}, m_{22}, m_{21}, m_{22})$	1061.38	0.1029	109.216002
$(m_{22}, m_{22}, m_{22}, m_{21})$	944.99	0.1029	97.239471
$(m_{22}, m_{22}, m_{22}, m_{22})$	931.94	0.2401	223.758794
Expected value			975.437

Table 7.7: Computing Q2′ under the by-tuple/expected value semantics

equivalent to its by-table counterpart. Before introducing this equivalence formally, we start with an illustrating example:

Example 79. *Consider query Q2′. Using the by-table/expected value semantics, we consider two possible cases. Using m_{21} we map price to bid, with a query outcome of* $195 + 200 + 331.94 + 349.99 = 1076.93$ *and a probability of* 0.3. *Using m_{22} we map price to currentPrice, with a query outcome of* $195 + 197.5 + 202.5 + 336.94 = 931.94$ *and a probability of* 0.7. *Therefore, the answer to Q2′, under the by-table/expected value semantics would be* $1076.93 * 0.3 + 931.94 * 0.7 = 975.437$.

Table 7.7 presents the 16 *different sequences and for each sequence it computes the query output, its probability, and the product of the two (which is a term in the summation defining expected value). The outcome of* Q2' *using the by-tuple/expected value semantics is identical to that of the by-table/expected value semantics. To see why, let us trace a single value,* 434.99*. This value appears in the fourth tuple and is used in the computation whenever a sequence contains mapping* m_{21} *for the fourth tuple, which is every other row in Table 7.7. Summing up the probabilities of all such worlds yields a probability of* 0.3*, which is exactly the probability of using* m_{21} *in the by-table semantics. The reason for this phenomenon is because the association of a mapping to one tuple is independent of the association with another tuple.*

Example 79 explains the intuition underlying Theorem 25 below. It is worth noting that this solution does not extend to the `AVG` aggregate because it is a non-monotonic aggregate.

Theorem 25. *Let* $\overline{pM} = (S, T, \mathbf{m})$ *be a schema p-mapping and let* Q *be a* `SUM` *query over attribute* $A \in S$*. The expected value of* $Q^{tuple}(D_T)$*, a* by-tuple *answer to* Q *with respect to* $\overline{pM}$*, is identical to* $Q^{table}(D_T)$*, a* by-table *answer to* Q *with respect to* $\overline{pM}$*.*

In order to prove this theorem, We will first prove a series of properties that are necessary to do so. The following notation will be used.

Notation. Let $\Pr(m)$ be the probability associated with mapping m. We order the $|\mathbf{m}|$ mappings and name them $m^{(1)}, m^{(2)}, ..., m^{(|\mathbf{m}|)}$. We denote by $A'_{i(k)}$ the value of the mapping of A of the i-th tuple using $m^{(k)}$. $\mathbf{seq}_{i(k)}\left(p\bar{M}\right)$ is the set of all sequences in which the i-th tuple uses the $m^{(k)}$ mapping.

Lemma 1. $\sum_{k=1}^{|\mathbf{m}|} \Pr(m^{(k)}) = 1$

Lemma 2. $\sum_{\text{seq} \in \mathbf{seq}\left(p\bar{M}\right)} \Pr\left(\text{seq}\right) = 1$

Proof. By induction on the number of mappings on $\mathbf{m}$.

Base: $|\mathbf{m}| = 1$. There is only one sequences, $\left|\mathbf{seq}\left(p\bar{M}\right)\right| = 1$ and $\Pr(m^{(j)}) = 1$ from Lemma 1. $\sum_{\text{seq}\in\mathbf{seq}\left(p\bar{M}\right)} \Pr\left(\text{seq}\right) = \prod_{j=1}^{|D_T|} \Pr(m^{(j)}) = \prod_{j=1}^{|D_T|} 1 = 1$

Step: Suppose that the induction assumption holds for $|\mathbf{m}| < q$. For $|\mathbf{m}| = q$ we partition the summation into $|\mathbf{m}|$ summations, each with all the sequences that share a common mapping for the first tuple:

$$\begin{aligned}
&\sum_{\text{seq}\in\mathbf{seq}\left(p\bar{M}\right)} \Pr\left(\text{seq}\right)\\
&= \Pr(m^{(1)}) \cdot \sum_{\text{seq}\in\mathbf{seq}_{1(1)}\left(p\bar{M}\right)} \prod_{j=2}^{|D_T|} \Pr(m_j)\\
&+ \Pr(m^{(2)}) \cdot \sum_{\text{seq}\in\mathbf{seq}_{1(2)}\left(p\bar{M}\right)} \prod_{j=2}^{|D_T|} \Pr(m_j)\\
&+ ...\\
&+ \Pr(m^{(|\mathbf{m}|)}) \cdot \sum_{\text{seq}\in\mathbf{seq}_{1(|\mathbf{m}|)}\left(p\bar{M}\right)} \prod_{j=2}^{|D_T|} \Pr(m_j)\\
&= \Pr(m^{(1)}) + \Pr(m^{(2)}) + ... + \Pr(m^{(|\mathbf{m}|)}) = 1
\end{aligned}$$

based on the induction assumption and Lemma 1. □

Lemma 3. $\sum_{\text{seq}\in\mathbf{seq}_{i(k)}\left(p\bar{M}\right)} \prod_{j=1}^{i-1} \Pr(m_j) \prod_{j=i+1}^{|D_T|} \Pr(m_j) = 1$

Proof. By induction on the number of tuples in T.

Base: $|\mathbf{D}_t| = 2$. In this case, the number of sequences in which one tuple keeps the same mapping is $|\mathbf{m}|$. Therefore,

$$\sum_{\text{seq}\in\mathbf{seq}_{i(k)}\left(p\bar{M}\right)} \prod_{j=1}^{i-1} \Pr(m_j) \prod_{j=i+1}^{|D_T|} \Pr(m_j)$$
$$= \Pr(m^{(1)}) + \Pr(m^{(2)}) + ... + \Pr(m^{(|\mathbf{m}|)}) = 1$$

from Lemma 1.

Step: Suppose that the induction assumption holds for $|\mathbf{D}_t| < q$. For $|\mathbf{D}_t| = q$ we choose a tuple different from the i-th tuple. Without loss of generality assume that we choose the first tuple. We partition the summation into $|\mathbf{m}|$ summations, each with all the sequences that share a common mapping for the first tuple:

$$\sum_{\text{seq}\in\mathbf{seq}_{i(k)}\left(p\bar{M}\right)} \prod_{j=1}^{i-1} \Pr(m_j) \prod_{j=i+1}^{|D_T|} \Pr(m_j)$$
$$= \Pr(m^{(1)}) \cdot \sum_{\text{seq}\in\mathbf{seq}_{i(k)\wedge 1(1)}\left(p\bar{M}\right)} \prod_{j=2}^{i-1} \Pr(m_j) \prod_{j=i+1}^{|D_T|} \Pr(m_j)$$
$$+ \Pr(m^{(2)}) \cdot \sum_{\text{seq}\in\mathbf{seq}_{i(k)\wedge 1(2)}\left(p\bar{M}\right)} \prod_{j=2}^{i-1} \Pr(m_j) \prod_{j=i+1}^{|D_T|} \Pr(m_j)$$
$$+ ...$$
$$+ \Pr(m^{(|\mathbf{m}|)}) \cdot \sum_{\text{seq}\in\mathbf{seq}_{i(k)\wedge 1(|\mathbf{m}|)}\left(p\bar{M}\right)} \prod_{j=2}^{i-1} \Pr(m_j) \prod_{j=i+1}^{|D_T|} \Pr(m_j)$$
$$= \Pr(m^{(1)}) + \Pr(m^{(2)}) + ... + \Pr(m^{(|\mathbf{m}|)}) = 1$$

based on the induction assumption and Lemma 1. □

Theorem 26. *Let $p\bar{M} = (S, T, \mathbf{m})$ be a schema p-mapping and let Q be a sum query over attribute $A \in S$. The expected value of $Q^{tuple}\,(D_S \cup D_T)$, a* by-tuple *answer to Q with respect to $p\bar{M}$, is identical to $Q^{table}\,(D_S \cup D_T)$, a* by-table *answer to Q with respect to $p\bar{M}$.*

Proof. Let $p\bar{M} = (S, T, \mathbf{m})$ be a p-mapping. Let Q be a sum query over attribute $A \in S$ and let D_S be an instance of S. First, let D_T be a by-table consistent instance of T. Therefore, there exists a mapping $m \in \mathbf{m}$ such that D_S and D_T satisfy m.

Given a mapping m, for which $A \in S$ is mapped to $A' \in T$, the outcome of Q is:

$$\sum_{i=1}^{|D_S|} A_i + \sum_{i=1}^{|D_T|} A'_i \tag{7.3}$$

The expected value of a *by-tuple* answer to Q with respect to $p\bar{M}$ is:

$$\begin{aligned}
&\sum_{j=1}^{|\mathbf{m}|}(\Pr(m^{(j)}) \cdot (\sum_{i=1}^{|D_S|} A_i + \sum_{i=1}^{|D_T|} A'_{i(j)}))\\
&= \sum_{j=1}^{|\mathbf{m}|}(\Pr(m^{(j)}) \cdot \sum_{i=1}^{|D_S|} A_i) + \sum_{j=1}^{|\mathbf{m}|}(\Pr(m^{(j)}) \cdot \sum_{i=1}^{|D_T|} A'_{i(j)})\\
&= \sum_{i=1}^{|D_S|} A_i \cdot \sum_{j=1}^{|\mathbf{m}|} \Pr(m^{(j)}) + \sum_{j=1}^{|\mathbf{m}|}(\Pr(m^{(j)}) \cdot \sum_{i=1}^{|D_T|} A'_{i(j)})\\
&= \sum_{i=1}^{|D_S|} A_i + \sum_{j=1}^{|\mathbf{m}|}(\Pr(m^{(j)}) \cdot \sum_{i=1}^{|D_T|} A'_{i(j)})
\end{aligned} \tag{7.4}$$

The justification for the move from the third to the fourth line is from Lemma 1.

Let's consider now a mapping sequence $\mathrm{seq} = \left\langle m^1, m^2, \ldots, m^{|D_T|} \right\rangle$. m^i can be one of $|\mathbf{m}|$ values. We can say, for example that $m^i = m^{(j)}$ which means that the mapping of the i-th tuple is using $m^{(j)}$ interpretation.

The associated probability sequence $\langle \Pr_1, \Pr_2, \ldots, \Pr_{|D_T|}\rangle$ assigns probability $\Pr_i \in \{\Pr(m_1), \Pr(m_2), \ldots, \Pr(m_{|\mathbf{m}|})\}$ to each tuple in D_T. Due to the independent assignment of interpretations to tuples, $\Pr(\mathrm{seq}) = \prod\limits_{i=1}^{|D_T|} \Pr(m^i)$. $\mathbf{seq}\left(p\bar{M}\right)$ is the set of all $m^{|D_T|}$ sequences that can be generated from $p\bar{M}$. Given a sequence $\mathrm{seq}_j \in \mathbf{seq}\left(p\bar{M}\right)$, we denote by m^{ij} the i-th element of the j-th sequence.

The expected value of a *by-tuple* answer to Q with respect to $p\bar{M}$ is:

$$
\begin{aligned}
&\sum_{\mathrm{seq}\in\mathbf{seq}\left(p\bar{M}\right)} (\Pr(\mathrm{seq}) \cdot \sum_{i=1}^{|D_S|} A_i) + \sum_{\mathrm{seq}\in\mathbf{seq}\left(p\bar{M}\right)} \Pr(\mathrm{seq}) \cdot \sum_{i=1}^{|D_T|} A'_i \\
&= \sum_{i=1}^{|D_S|} A_i \cdot \sum_{\mathrm{seq}\in\mathbf{seq}\left(p\bar{M}\right)} \Pr(\mathrm{seq}) + \sum_{k=1}^{\left|\mathbf{seq}\left(p\bar{M}\right)\right|} \prod_{i=1}^{|D_T|} Pr(m^{ij}) \cdot \sum_{i=1}^{|D_T|} A'_i \\
&= \sum_{i=1}^{|D_S|} A_i + \sum_{k=1}^{\left|\mathbf{seq}\left(p\bar{M}\right)\right|} \prod_{i=1}^{|D_T|} Pr(m^{ij}) \cdot \sum_{i=1}^{|D_T|} A'_i
\end{aligned}
\tag{7.5}
$$

For a given i and j, let's have a look now at all the sequences in which $A'_{i(j)}$ appear. From the construction of the sequence set, there are exactly $\frac{1}{|\mathbf{m}|}$ such sequences and they all share in common that $m^{ik} = m^{(j)}$, since $A'_{i(j)}$ uses mapping $m^{(j)}$. This part of the summation can be written to be

$$
\begin{aligned}
&\Pr(m^{(j)}) \cdot A'_{i(j)} \cdot \sum_{\mathrm{seq}\in\mathbf{seq}_{i(j)}\left(p\bar{M}\right)} \prod_{k=1}^{i-1} \Pr(m^{ik}) \prod_{k=i+1}^{|D_T|} \Pr(m^{ik}) \\
&= \Pr(m^{(j)}) \cdot A'_{i(j)}
\end{aligned}
$$

We repeat this computation for all $A'_{i(j)}$ and Eq 7.5 can now be rewritten to be

$$\sum_{i=1}^{|D_T|}\sum_{j=1}^{|\mathbf{m}|} A'_{i(j)} \cdot \Pr(m^{(j)})$$
$$= \sum_{j=1}^{|\mathbf{m}|}(Pr(m^{(j)}) \cdot \sum_{i=1}^{|D_T|} A'_{i(j)})$$

Adding the sum of the A attribute in the S table one has that the expected value of a *by-tuple* answer to Q with respect to $p\bar{M}$ is:

$$= \sum_{i=1}^{|D_S|} A_i + \sum_{j=1}^{|\mathbf{m}|}(Pr(m^{(j)}) \cdot \sum_{i=1}^{|D_T|} A'_{i(j)})$$

Corollary 7. *The expected value of a* by-tuple *answer to a query Q with respect to a schema p-mapping pM is PTIME with respect to data complexity and mapping complexity.*

□

Aggregate functions MAX and MIN

We now present an efficient algorithm to compute the `MAX` aggregate under the range semantics for the by-tuple semantics. The techniques presented here for `MAX` can be easily adapted for answering queries involving the `MIN` aggregate.

`MAX` *under the Range semantics.* To compute `MAX` under the range semantics, we have to find the minimum and the maximum value of the aggregate under any possible mapping sequence, *i.e.*, the tightest interval that includes all the possible maximum values that can arise. The procedure to find this interval without the need to look at all possible sequences is outlined in Figure 7.5.

```
Algorithm ByTupleRangeMAX
Input: Table $S, T$; MapList $M$; Attribute $A$; Condition $C$;

For each $t_i \in S$,
    let $v_i^{min}$ be the minimum value obtained by applying a mapping in $M$
    that satisfies condition $C$;
    Similarly, let $v_i^{max}$ be the maximum value.
return $[\max_i\{v_i^{min}\}, \max_i\{v_i^{max}\}]$;
```

Figure 7.5: Algorithm to answer `SELECT MAX(A) FROM T` under Range Semantics

To see how this algorithm works, consider Example 76. We answer the subquery within the FROM clause of query Q2:

`SELECT MAX(DISTINCT` T2.price) `FROM` T2 `AS` R2 `GROUP BY` R2.auctionID

This subquery contains a `GROUP BY` auctionID, which means we will have one answer for each distinct auctionID. In this case, looking at Table 7.2 we see that the answer will consist of two different ranges, one for auctionID $= 34$ and another for auctionID $= 38$. We show how to compute the answer for auctionID $= 38$; The process to obtain the answer for auctionID $= 34$ is analogous. For tuple 5, with transactionID $= 3801$, the minimum value obtained by applying a mapping is $v_5^{min} = 300$, while the maximum is $v_5^{max} = 330.01$. For tuple 6, $v_6^{min} = 335.01$ and $v_6^{max} = 429.95$; for tuple 7, $v_7^{min} = 336.3$ and $v_7^{max} = 439.95$. Finally, for tuple 8, $v_8^{min} = 340.05$ and $v_8^{max} = 438.05$. Now, the range for the aggregator is given by $[\max_i\{v_i^{min}\}, \max_i\{v_i^{max}\}]$. Where each bound is computed as:

$$\max_i\{v_i^{min}\} = \max\{300, 335.01, 336.3, 340.05\}$$

$$\max_i\{v_i^{max}\} = \max\{330.01, 429.95, 439.95, 438.05\}$$

and thus, the final answer is $[340.05, 439.95]$. In general, it is always the case that the range yielded by the by-table semantics is a subset of the range yielded by the by-tuple semantics. This is because by-tuple has the possibility of choosing a different mapping for each tuple, which means that the algorithm has the freedom to choose sequences that are not allowed using the by-table semantics. This is true for all aggregate functions considered in this work. This algorithm also requires a polynomial number of computations in $O(m * n)$, where m is the number of mappings and n is the number of tuples.

Theorem 27. *Algorithm `ByTupleRangeMAX` correctly computes the result of executing an* `MAX` *query under the by-tuple/range semantics.*

Proof. Suppose, towards a contradiction, that the algorithm returns the range $[\ell, u]$ and that there exists a possible answer k such that either $k < \ell$ or $u < k$. If k is in fact a possible value for the MAX query, this means that there is at least one tuple in T such that, for some mapping, the value of A is less (respectively, greater) than v_i^{min} (respectively, v_i^{max}), which is a contradiction since the algorithm chooses this value to be the minimum (respectively, maximum) under all possible mappings. □

7.4.3 Summary of Complexity Results

The tables in Figure 7.6 are a summary of our results for the six different kinds of semantics. The algorithms presented in this section correspond to those that require polynomial time to compute the answer.

7.5 Experimental Results

In order to evaluate the difference in the running times of our algorithms (both PTIME and non-PTIME), and how these are affected by changes in both the number of tuples in the database and the number of probabilistic mappings present, we carried out a series of empirical tests whose results we report in this section. The algorithms we

COUNT	Range	Distribution	Expected Value
By-Table	PTIME	PTIME	PTIME
By-Tuple	PTIME	PTIME	PTIME

SUM	Range	Distribution	Expected Value
By-Table	PTIME	PTIME	PTIME
By-Tuple	PTIME	?	PTIME

MAX,MIN,AVG	Range	Probability Distribution	Expected Value
By-Table	PTIME	PTIME	PTIME
By-Tuple	PTIME	?	?

Figure 7.6: Summary of complexity for the different aggregates

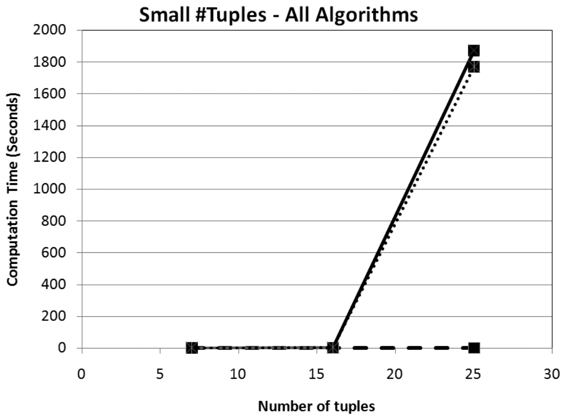


Figure 7.7: Running times for variation of #tuples using the eBay data; #attributes = 7, #mappings = 2, results are averages over 5 runs on the eBay auction data. *Solid line*: ByTuplePDMAX. *Dotted line*: ByTupleExpValAVG, ByTuplePDAVG, ByTuplePDSUM, and ByTupleExpValMAX. *Dashed line (touching the x axis)*: ByTupleRangeMAX, ByTupleRangeCOUNT, ByTuplePDCOUNT, ByTupleExpValCOUNT, ByTupleRangeSUM, ByTupleExpValSUM, and ByTupleRangeAVG.)

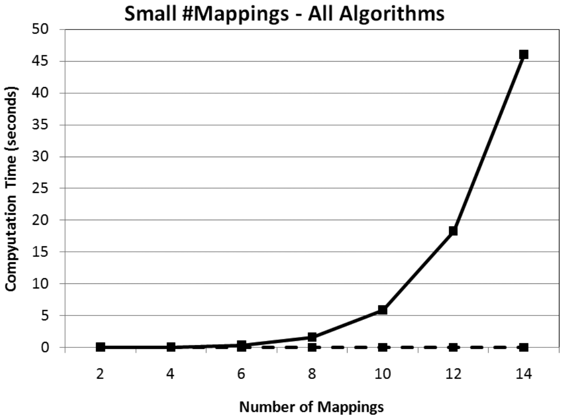


Figure 7.8: Running times for variation of #mappings; #attributes = 20, #tuples = 6, results are averages over 5 runs on synthetic data. *Solid line*: ByTupleExpValAVG, ByTuplePDAVG, ByTuplePDSUM, ByTupleExpValMAX, and ByTuplePDMAX. *Dashed line (touching the x axis)*: ByTupleRangeMAX, ByTupleRangeCOUNT, ByTuplePDCOUNT, ByTupleExpValCOUNT, ByTupleRangeSUM, ByTupleExpValSUM, and ByTupleRangeAVG.

gave for problems that were not shown to be PTIME are — as expected — inefficient. However, the algorithms we gave for problems we showed to be in PTIME are quite efficient when we vary both the number of tuples and the numbers of mappings — but clearly there are limits that vary from one algorithm to another. We will discuss these limits below.

The programs to carry out these tests consist of about 3,300 lines of Java code. All computations were carried out on a quad-processor computer with Intel Xeon 5140 dual core CPUs at 2.33GHz each, 4GB of RAM, under the CentOS GNU/Linux OS (distribution 2.6.9-55.ELsmp). The database engine we used was PostgreSQL version 7.4.16.

Experimental Setup. We carried out two sets of experiments. The first set used *real-world data* of 1,129 eBay 3-day auctions with a total of 155,688 bids for Intel, IBM, and Dell laptop computers. The data was obtained from an RSS feed for a search query on eBay.[1] The database schema is the one presented in Example 76. The sole point of

[1] http://search.ebay.com/ws/search/

uncertainty lies in the two price attributes where a reference to Price could mean either the bid price or the current price. We therefore defined two mappings: bid mapped to Price with probability 0.3 and currentPrice mapped to Price with probability 0.7. We have applied the inner query of query Q2 and also a set of queries that cover four different operators discussed in this work (all except MIN).

The second set of experiments was done on synthetic, randomly generated data in order to be able to evaluate configurations not possible with the eBay data (in particular, larger numbers of attributes, tuples, and mappings). The tables consist of attributes of type real, plus one column of type int used as id (not included in the number of attributes reported in the results). Mappings were also randomly generated by selecting an attribute at random and then a set of attributes that are mapped to it, also with a randomly chosen probability distribution. Each experiment was repeated several times.

Results. We now present and analyze the experiment results for small, medium, and large instances.

Small instances. We ran a set of experiments on small relations to compare the performance of all possible semantics, including those for which there are no PTIME algorithms. Figures 7.7 and 7.8 show the running times of all algorithms on small instances (#mappings fixed at 2 in the former, #tuples fixed at 6 in the latter). The former corresponds to runs using the eBay auction data (results shown on a scatterplot, since each point corresponds to adding all tuples from an auction), while the latter reports results from runs on synthetic data.

As we can see, running times climb exponentially for algorithms we did not show to be in PTIME; the sharp increase in Figure 7.7 continues when more auctions are included, with a completion time of more than 10 days for 4 auctions (36 tuples). On the other hand, the running times of the other algorithms are negligible. When we varied #tuples,

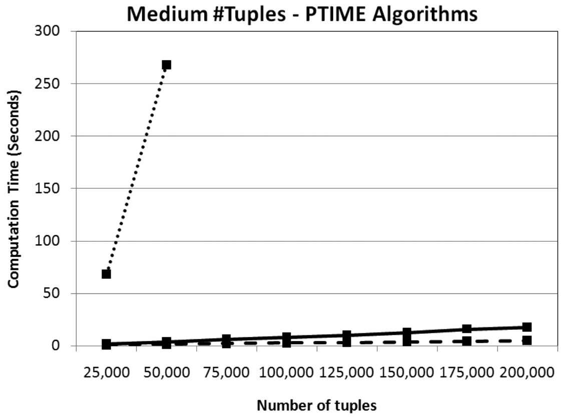


Figure 7.9: Running times for variation of #tuples; #attributes = 50, #mappings = 20, results are averages over 5 runs on synthetic data. *Solid line*: ByTupleRangeAVG, ByTupleRangeSUM, ByTupleRangeCOUNT, and ByTupleRangeMAX. *Dashed line*: ByTupleExpValSUM. *Dotted line*: ByTuplePDCOUNT and ByTupleExpValCOUNT.

the by-table algorithms running times lay between 0.07 and 0.13 seconds. When we varied #mappings, the by-table algorithms took between 0.03 and 0.26 seconds. We also ran experiments varying #tuples using synthetic data which yielded the same trends in running times as those in Figure 7.7.

Medium-size instances. Figure 7.9 shows the running times of all our PTIME algorithms when the number of tuples is increased into the tens and hundreds of thousands (#mappings fixed at 20). As we can see, the `ByTuplePDCOUNT` and `ByTupleExpValCOUNT` algorithms' performance is well differentiated from the rest, as they become intractable at about 50,000 tuples. This is due to the fact that these algorithms must update the probability distribution for the possible values in each iteration, leading to a running time in $O(m * n^2)$ as shown in Section 7.4.2. In this case, the by-table algorithms' running times varied between 0.96 seconds and 5.49 seconds.

Figure 7.10 shows how the running times increase with the number of mappings (#tuples fixed at 50,000, #attributes = 500). Note that `ByTupleExpValSUM` is more affected by the increase in number of mappings than the other four algorithms, with its running time climbing to almost 90 seconds for 250 mappings. This is because it is a

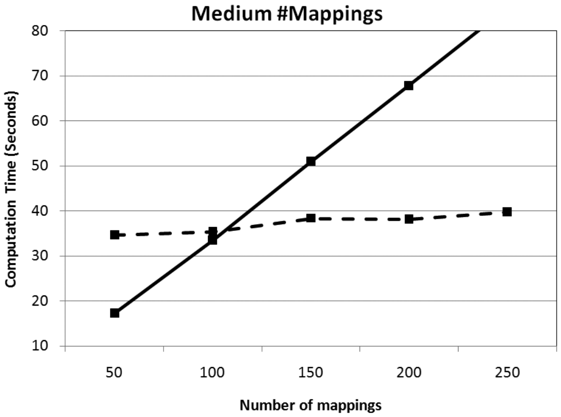


Figure 7.10: Running times for variation of #mappings; #attributes = 500, #tuples = 50,000, results are averages over 2 runs on synthetic data. *Solid line*: ByTupleExpValSUM. *Dashed line*: ByTupleRangeMAX, ByTupleRangeCOUNT, ByTupleRangeSUM, and ByTupleRangeAVG.

by-table algorithm, and it must issue as many queries as mappings and then combine the answers. The other four, on the other hand, only slightly increase their running times at these numbers of mappings. The by-table algorithms' running times in this case lie between 16.49 and 86.49 seconds.

Large instances. Figure 7.11 shows how our most scalable by-tuple algorithms perform when the number of tuples is increased into the millions, showing that algorithms `ByTupleRangeMAX/COUNT/AVG/SUM` take about 1,300 seconds (about 21 minutes) to answer queries with 5 million tuples and 20 mappings. This figure also shows the running time of `ByTupleExpValSUM`, which is much lower than the others because it is actually equivalent to the by-table algorithm, as seen in Section 7.4.2. The corresponding running times for the by-table algorithms varied between 15.73 and 125.63 seconds. We also ran experiments for 15 to 30 million tuples, the results of which are shown in Figure 7.12. For these runs, the by-table algorithms took between 65.17 seconds and 124.76 seconds.

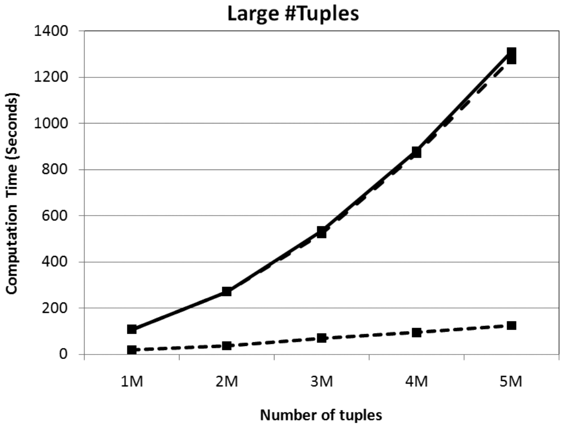


Figure 7.11: Running times for variation of #tuples; #attributes = 50, #mappings = 20, results are averages over 2 runs on synthetic data. *Solid line*: ByTupleRangeMAX and ByTupleRangeAVG. *Dashed line*: ByTupleRangeSUM and ByTupleRangeCOUNT. *Dotted line*: ByTupleExpValSUM.

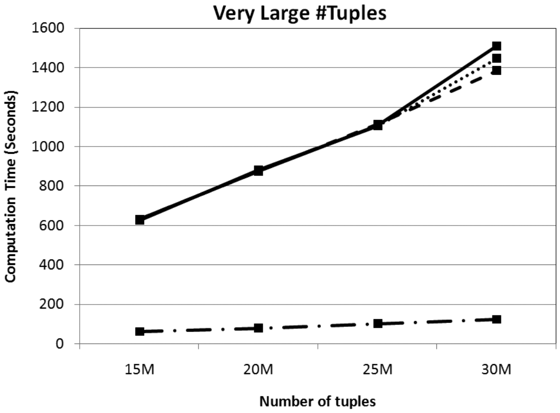


Figure 7.12: Running times for variation of #tuples; #attributes = 20, #mappings = 5, results are averages over 2 runs on synthetic data. *Solid line*: ByTupleRangeCOUNT. *Dotted line*: ByTupleRangeSUM and ByTupleRangeAVG. *Dashed line*: ByTupleRange-MAX. *Dashed and Dotted line*: ByTupleExpValSUM.

It should be noted that the greater scalability of the by-table algorithms with respect to the efficient by-tuple algorithms presented here is in large part due to the fact that the former are taking advantage of the optimizations implemented by the DBMS when answering queries.

7.6 Schema Mappings, Integrity Constraints, and Partial Information

In this chapter we have defined and analyzed different semantics for aggregate query answering in the setting of data integration using probabilistic schema mappings. An important assumption we have made across this chapter is that the data is both complete and consistent, problems in data integration only arise from the uncertainty at schema level. In real world data sets this is often not the case, and it is therefore a good idea to investigate the problem of query answering under probabilistic schema mappings in the presence of inconsistent and partial information. Other works in the literature have studied the relationship between schema mappings and integrity constraints. The iMAP [DLD$^+$04] system, for instance, exploits integrity constraints in order to define meaningful mappings among disparate schemas and to prune the search space of the mapping generation process. Some constraint may state, for instance, that two attributes are unrelated and therefore they should not appear together in a mapping. Other constraints such as functional dependencies or denial constraints can be used as well; however, depending on the complexity of the constraint it might be too expensive to check it in the data in order to avoid considering possibilities that violate them. It is important to note that the use of integrity constraints in generating mappings does not guarantee that the actual integrated data is going to remain consistent, since posterior updates or insertions to any of the sources tables may introduce inconsistency with respect to the set of integrity constraints.

Consider the source and target schemas for the running example in this chapter:

S1 = (ID, price, agentPhone, postedDate, reducedDate)

T1 = (propertyID, listPrice, phone, date, comments)

The following table represents the instance relation D_{S1}.

ID	price	agentPhone	postedDate	reducedDate
1	100k	215	1/5/2008	2/5/2008
2	150k	342	1/30/2008	2/15/2008
3	200k	215	1/1/2008	1/10/2008
4	100k	337	1/2/2008	2/1/2008

Now suppose that in addition to D_{S1} we also have an instance relation for T1, *i.e.*, there exist data that corresponds to the target schema. Let D_{T1} consist of the following tuples.

propertyID	listPrice	phone	date	comments
1	100k	215	1/5/2008	c_1
2	150k	342	1/30/2008	c_2

As before, possible mappings between S1 and T1 are: m_{11} where date maps to postedDate and m_{12} where date maps to reducedDate. In addition, suppose that we have the integrity constraint $fd : propertyID \to listPrice$ over T1.

A tool such as iMAP could rule out any of the two mappings if when applied to D_{S1} and D_{T1} the result violates fd. However, in these instance relations no conflict can appear w.r.t. fd and therefore both m_{11} and m_{12} are considered possible mappings.

Now, suppose that S1 is updated with some new information in the following way:

propertyID	listPrice	phone	date	comments
1	120k	215	1/5/2008	c_1
2	150k	342	1/30/2008	c_2

If we were to issue the query "Select $propertyID, listPrice$ from T where $listPrice < 200K$ and $1/1/2008 < date < 1/30/2008$", depending on which semantics (by-table or by-tuple), the result might violate fd.

The point of this example was to show that inconsistency can appear in answers to queries as well as in the result of using schema mapping methods, even when each source is consistent in isolation.

Alternatively, works such those from [GN06, FKMP05, FKP05] have analyzed data exchange when tuple generating (TGDs) and equality generating dependencies (EDGs) are considered. Note that because of the presence of TGDs, null values may appear and therefore it is necessary to deal with them. The idea in those approaches is to focus on a class of solutions, called *universal solutions*, possessing desirable properties that justify selecting them as the semantics of the data exchange problem. Universal solutions have the property that homomorphisms can be defined between them and every possible solution, and any pair of universal solutions is homomorphically equivalent. Universal solutions are the most general among all solutions and they represent the entire space of solutions, all universal solutions share a unique (up to isomorphism) common part, called their *core*. The core of a structure is the smallest substructure that is also a homomorphic image of the structure. Computing the core is a hard problem, but when restricted only to EDGs [FKMP05], or a combination of EGDs and weakly acyclic TGDs (or other cases where the chase procedure is known to terminate) [GN06], it can be computed in polynomial time in the data complexity. The problem of query answering in this setting was studied in [FKP05]. In that work, universal solutions are used to compute the *certain answers* of queries q that are unions of conjunctive queries over the target schema. The set $certain(q, I)$ of certain answers of a query q over the target schema, with respect to a source instance I, consists of all tuples that are in the intersection of all $q(J)$'s, as J varies over all solutions for I ($q(J)$ denotes the result of evaluating query q on J). Given I and J, $certain(q, I)$ is the set of all "null-free" tuples in $q(J)$. The set $certain(q, I)$ is computable in time polynomial in the cardinality of I if q is a union of conjunctive queries over the target schema: first compute a universal J solution in polynomial time and then evaluate $q(J)$ and remove tuples with nulls. However, if q is a union of conjunctive queries with at most two inequalities per conjunct, then the problem is coNP-complete. Alternatively, [FKP05] defined *universal certain answers*; $ucertain(q, I)$ consists of all tuples that are in the intersection of all $q(J)$'s, as J varies over all universal solutions for I. The set $ucertain(q, I)$ can be computed in polynomial time for existential queries whenever the core of I can be computed in polynomial time. This approach was developed for and implemented in the CLIO system [FHH$^+$09].

The approach described above provides a fixed semantics for inconsistent and incompleteness resolution. As clearly stated in the definition of certain answers, the set of tuples in the answer is the set of all tuples that are free of null values across all possible (valid) ways of completing the database instance. Furthermore, if instance I is inconsistent, the chase procedure used to compute the universal solutions for I fails and no universal solution is computed; therefore, the empty set is returned as the answer to the query. Finally, probabilistic schema mappings are not explored in any of these works. Schema mappings are represented as high level constraints using source-to-target tuple generating dependencies (see Chapter 2.4). Alternatively, we propose a personalizable approach by incorporating IMPs, described in Chapter 3, and PIPs, described in Chapter 6, in order to handle inconsistency and partial information, respectively. We consider two different options to solve this problem. The simplest one is to perform a post-query process. IMPs and PIPs can both be specified to be applied after the query is answered. In the presence of uncertain schema mappings, each tuple in the answer will have associated a probability; in these cases, IMPs and PIPs can be built to make use of these probabilities in order to manage the inconsistency or incompleteness following a certain strategy. Alternatively, a second approach would be to embed IMPs and/or PIPs in the query algorithms for both by-table and by-tuple semantics. For the latter we need to investigate the role that the policies play in obtaining the possible answers. We leave this task as future work.

7.7 Concluding Remarks

Probabilistic schema matching is emerging as a paradigm for integrating information from multiple databases. In past work, [DHY07] has proposed two semantics called the by-table and by-tuple semantics for selection, projection and join query processing under probabilistic schema mapping.

In this chapter, we studied the problem of answering aggregate queries in such an environment. We present three semantics for aggregates — a range semantics in which a range of possible values for an aggregate query is returned, a probability distribution semantics in which all possible answer values are returned together with their probabilities, and an expected value semantics. These three semantics combine together with the semantics of [DHY07] to provide six possible semantics for aggregates. Given this setting, we provide algorithms to answer `COUNT`, `SUM`, `AVG`, `MIN`, and `MAX` aggregate queries.

We develop algorithms for each of these five aggregate operators under each of the six semantics. The good news is that for every aggregate operator, at least one (and sometimes more) semantics are PTIME computable.

We also report on a prototype implementation and experiments with two data sets — a real world eBay data set, and a synthetic data set. We experimentally show that each aggregate operator listed above can be computed efficiently in at least one of these six semantics, even when the data set is large and there are many different possible schema mappings.

Chapter 8

Conclusions

In this thesis, we have proposed several frameworks for dealing with problems that arise from data or knowledge integration. The approach of our proposals is different from previous attempts, both in artificial intelligence and databases, since it focuses on the prior knowledge and expectations of the data and the application that the actual user can bring into the data management processes, giving him the power to do it in the way that best suits his needs.

After introducing real world scenarios in which data integration issues arise in Chapter 1, and reviewing the literature that is most closely related to this work in Chapter 2, in Chapter 3 we proposed a policy based framework for personalizable management of inconsistent information that allows users to bring their application expertise to bear. We define formally the notion of *inconsistency management policies*, or IMPs, with respect to functional dependencies as functions satisfying a *minimal* set of axioms. We proposed several families of IMPs that satisfy these axioms, and study relations between them in the simplified case where only one functional dependency is present. We show that when multiple functional dependencies are considered, multiple alternative semantics can result. We introduced new versions of the relational algebra that are augmented by inconsistency management policies that are applied either before the operator or after. We develop theoretical results on the resulting extended relational operators that could, in principle, be used in the future as the basis of query optimization techniques. Finally, we presented an index structure for implementing an IMP-based framework showing that it is versatile, can be implemented based on the needs and resources of the user and, according to our theoretical results, the associated algorithms incur reasonable costs. As a consequence, IMPs are a powerful tool for end users to express what they wish to do with their data, rather than have a system manager or a DB engine that does not understand their domain problem dictate how they should handle inconsistencies in their data.

In Chapter 4 we developed a general and unified framework for reasoning about inconsistency in a wide variety of monotonic logics. The basic idea behind this framework is to construct what we call *options*, and then using a preference relation defined by the user to compute the set of *preferred options*, which are intended to support the conclusions to be drawn from the inconsistent knowledge base. We provide a formal definition of the framework as well as algorithms to compute preferred options. We have also shown through examples how this abstract framework can be used in different logics, provided new results on the complexity of reasoning about inconsistency in such logics, and proposed general algorithms for computing preferred options. Furthermore, we showed that our general framework allows to represent approaches to inconsistency that are well-known in the artificial intelligence literature.

Focusing on more specific domains, in Chapter 5 we develop a formalism for identifying inconsistencies in news reports. Besides the complication of having to deal with a very large number of records, collecting and analyzing records such as news reports encounters an extra level of complexity: the presence of linguistically modified terms that can be interpreted in different ways and make the notion of inconsistency less clear. We propose a probabilistic logic programming language called PLINI within which users can write rules specifying what they mean by inconsistency.

In Chapter 6, we proposed the concept of a *partial information policy* (PIP). Using PIPs, end-users can specify the policy they want to use to handle partial information. We presented examples of three families of PIPs that end-users can apply. Many more are possible as simple combinations of these basic PIPs – in addition, the definition of PIPs allows many more policies to be captured. We have also presented index structures for efficiently applying PIPs, and conducted an experimental study showing that the adoption of such index structures allows us to efficiently manage large data sets. Moreover, we have shown that PIPs can be combined with relational algebra operators, giving even more capabilities to users on how to manage their incomplete data. In previous works dealing with the management of incomplete databases, the DBMS dictates how incomplete information should be handled.

Finally, in Chapter 7 we analyzed the problem of how to answer aggregate queries in the presence of uncertain schema mappings. Two semantics had been proposed in the literature for answering SPJ queries in the presence of probabilistic schema mappings [DHY07]. In this chapter we proposed three semantics for aggregate query answering: a range semantics in which a range of possible values for an aggregate query is returned, a probability distribution semantics in which all possible answer values are returned together with their probabilities, and an expected value semantics. These three semantics combine together with the semantics of [DHY07] to provide six possible semantics for aggregates. Given this setting, we provide algorithms to answer `COUNT`, `SUM`, `AVG`, `MIN`, and `MAX` aggregate queries. We develop algorithms for each of these five aggregate operators under each of the six semantics. The good news is that for every aggregate operator, at least one (and sometimes more) semantics are PTIME computable.

Recently, researchers have begun to understand that uncertainty, in particular inconsistency and partial information, does not always need to be eliminated; it is possible and, more often than not, necessary to reason with an inconsistent and/or partial knowledge base. We need to use inconsistent and partial information; sometimes this kind of information can help to better understand the data and to improve the quality of decision making processes. Currently, there is a large gap between methodologies in artificial intelligence and databases, both for knowledge management and what is available to real users. There is a need for context-sensitive data management approaches that create synergy with the user in order to be useful. In this thesis we aimed towards bridging this gap by proposing several frameworks that provide personalizable approaches to the problems that arise from data integration and management.

Bibliography

[ABC99] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[ABC03a] M. Arenas, L. E. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[ABC$^+$03b] M. Arenas, L. E. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theoretical Computer Science*, 3(296):405–434, 2003.

[AC02] L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *AMAI*, 34(1):197–215, 2002.

[AFM06] Periklis Andritsos, Ariel Fuxman, and Renee J. Miller. Clean answers over dirty databases: A probabilistic approach. In *International Conference on Data Engineering (ICDE)*, page 30, Washington, DC, USA, 2006. IEEE Computer Society.

[AG85] Serge Abiteboul and Gösta Grahne. Update semantics for incomplete databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 1–12. VLDB Endowment, 1985.

[AGM85] Carlos E. Alchourron, Peter Gardenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2):510–530, 1985.

[AKG91] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.

[AKK$^+$03] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. Miller, and J. Mylopoulos. The hyperion project: From data integration to data coordination. *SIGMOD Record*, 32(3), 2003.

[AM86] Paolo Atzeni and Nicola M. Morfuni. Functional dependencies and constraints on null values in database relations. *Information and Control*, 70(1):1–31, 1986.

[AMB$^+$11] Massimiliano Albanese, Maria Vanina Martinez, Matthias Broecheler, John Grant, and V.S. Subrahmanian. Plini: a probabilistic logic program framework for inconsistent news information. *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, LNCS*, 6565, 2011.

[AP07] L. Amgoud and H. Prade. Formalizing practical reasoning under uncertainty: An argumentation-based approach. In *IAT*, pages 189–195, 2007.

[Art08] A. Artale. Formal methods: Linear temporal logic, 2008.

[AS07] Massimiliano Albanese and V. S. Subrahmanian. T-REX: A domain-independent system for automated cultural information extraction. In *International Conference on Computational Cultural Dynamics (ICCCD)*, pages 2–8. AAAI Press, August 2007.

[BB03] P. Barceló and L. E. Bertossi. Logic programs for querying inconsistent databases. In *PADL*, pages 208–222, 2003.

[BBC04] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1):89–113, 2004.

[BBFL05] L. E. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. Complexity and approximation of fixing numerical attributes in databases under integrity constraints. In *DBPL*, pages 262–278, 2005.

[BC03] L. E. Bertossi and J. Chomicki. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2003.

[BCD$^+$93] Salem Benferhat, Claudette Cayrol, Didier Dubois, Jérôme Lang, and Henri Prade. Inconsistency management and prioritized syntax-based entailment. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 640–647, 1993.

[BD83] Dina Bitton and David J. DeWitt. Duplicate record elimination in large data files. *ACM Transactions on Database Systems*, 8(2):255–265, 1983.

[BDKT97] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

[BDP97] S. Benferhat, D. Dubois, and Henri Prade. Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1: The flat case. *Studia Logica*, 58(1):17–45, 1997.

[BDSH$^+$08] Omar Benjelloun, Anish Das Sarma, Alon Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2):243–264, 2008.

[Bel77] N. Belnap. A useful four valued logic. *Modern Uses of Many Valued Logic*, pages 8–37, 1977.

[BFFR05] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[BG04] Indrajit Bhattacharya and Lise Getoor. Iterative record linkage for cleaning and integration. In *Workshop on Research issues in data mining and knowledge discovery*, pages 11–18, New York, NY, USA, 2004. ACM.

[BG07] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), 2007.

[BGMM$^+$09] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *VLDB Journal*, 18(1):255–276, 2009.

[BGMP92] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 4(5):487–502, 1992.

[BH05] Philippe Besnard and Anthony Hunter. Practical first-order argumentation. In *Conference on Artificial Intelligence (AAAI)*, pages 590–595, 2005.

[Bis79] Joachim Biskup. A formal approach to null values in database relations. In *Advances in Data Base Theory*, pages 299–341, 1979.

[BKM91] C. Baral, S. Kraus, and J. Minker. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 3(2):208–220, 1991.

[BKMS91] C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining knowledge bases consisting of first order theories. pages 92–101, 1991.

[BM03] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *International conference on Knowledge discovery and data mining (KDD)*, pages 39–48, New York, NY, USA, 2003. ACM.

[BMP$^+$08] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema mapping verification: The spicy way. In *EDBT*, 2008.

[Bob80] D. G. Bobrow. Special issue on non-monotonic reasoning. *Artificial Intelligence*, 13 (1-2), 1980.

[Bre89] G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1043–1048, 1989.

[BS89] H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68(2):135–154, 1989.

[BS98] P. Besnard and T. Schaub. Signed systems for paraconsistent reasoning. *Journal of Automated Reasoning*, 20(1):191–213, 1998.

[BV08] N. Bozovic and V. Vassalos. Two-phase schema matching in real world relational databases. In *International Conference on Data Engineering (ICDE)*, pages 290–296, 2008.

[CA03] G. Chen and T. Astebro. How to deal with missing categorical data: Test of a simple bayesian method. *Organ. Res. Methods*, 6(3):309–327, 2003.

[CGS97a] K. Selçuk Candan, John Grant, and V. S. Subrahmanian. A unified treatment of null values using constraints. *Information Sciences*, 98(1-4):99–156, 1997.

[CGS97b] K. Selçuk Candan, John Grant, and V. S. Subrahmanian. A unified treatment of null values using constraints. *Information Sciences*, 98(1-4):99–156, 1997.

[CGZ09] Luciano Caroprese, Sergio Greco, and Ester Zumpano. Active integrity constraints for database consistency maintenance. *IEEE Transactions on Knowledge Data Engineering (TKDE)*, 21(7):1042–1058, 2009.

[Cho07] J. Chomicki. Consistent query answering: Five easy pieces. In *International Conference on Database Theory (ICDT)*, pages 1–17, 2007.

[CKP03] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, New York, NY, USA, 2003. ACM.

[Cla77] K. L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322, 1977.

[CLR03] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 260–271, 2003.

[CLS94] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. On the complexity of non-monotonic entailment in syntax-based approaches. In *ECAI workshop on Algorithms, Complexity and Commonsense Reasoning*, 1994.

[CLS95] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Non-monotonic syntax-based entailment: A classification of consequence relations. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, pages 107–114, 1995.

[CM05] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005.

[Cod74] E. F. Codd. Understanding relations. *SIGMOD Records*, 6(3):40–42, 1974.

[Cod79] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

[CP87] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 71–81, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[CR02] William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *International conference on Knowledge discovery and data mining (KDD)*, pages 475–480, New York, NY, USA, 2002. ACM.

[CR08] A. G. Cohn and J. Renz. Qualitative spatial representation and reasoning. In F. van Hermelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, pages 551–596. Elsevier, 2008.

[CS86] H. Y. Chiu and J. Sedransk. A bayesian procedure for imputing missing values in sample surveys. *J. Amer. Statist. Assoc.*, 81(3905):5667–5676, 1986.

[CS00] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines*. Cambridge university press, 2000.

[CSD$^+$08] X. Chai, M. Sayyadian, A. Doan, A. Rosenthal, and L. Seligman. Analyzing and revising mediated schemas to improve their matchability. In *International Conference on Very Large Data Bases (VLDB)*, Auckland, New Zealand, August 2008.

[dC74] N.C.A. da Costa. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic*, 15(4):497–510, 1974.

[DHY07] Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *International Conference on Very Large Data Bases (VLDB)*, pages 687–698, 2007.

[DI03] E. D. Demaine and N. Immorlica. Correlation clustering with partial information. *Lecture Notes in Computer Science*, pages 1–13, 2003.

[DLD$^+$04] Robin Dhamankar, Yoonkyong Lee, Anhai Doan, Alon Halevy, and Pedro Domingos. imap: discovering complex semantic matches between database schemas. In *SIGMOD*, pages 383–394, New York, NY, USA, 2004. ACM.

[DNH04] A. Doan, N. F. Noy, and A. Y. Halevy. Introduction to the special issue on semantic integration. *SIGMOD Record*, 33(4):11–13, 2004.

[DS07] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, 2007.

[DSDH08] A. Das Sarma, X. Dong, and A.Y. Halevy. Bootstrapping pay-as-you-go data integration systems. pages 861–874, 2008.

[Dun95] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, Volume 77:pp. 321–357, 1995.

[Eme90] E. A. Emerson. Temporal and modal logic. In *Theoretical Computer Science*, pages 995–1072. 1990.

[ESS05] M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with apfel. In *International Semantic Web Conference (ISWC)*, pages 186–200, 2005.

[FFM05] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD*, pages 155–166, 2005.

[FFP05] S. Flesca, F. Furfaro, and F. Parisi. Consistent query answers on numerical databases under aggregate constraints. In *DBPL*, pages 279–294, 2005.

[FFP07] S. Flesca, F. Furfaro, and F. Parisi. Preferred database repairs under aggregate constraints. In *SUM*, pages 215–229, 2007.

[FH94] Ronald Fagin and Joseph Y. Halpern. Reasoning about knowledge and probability. *Journal of the ACM*, 41(2):340–367, 1994.

[FHH$^+$09] Ronald Fagin, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236, 2009.

[FHM90] R. Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1-2):78–128, 1990.

[Fit91] M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(1-2):91–116, 1991.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[FKP05] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Transactions on Database Systems*, 30(1):174–210, 2005.

[FKUV86] Ronald Fagin, Gabriel M. Kuper, Jeffrey D. Ullman, and Moshe Y. Vardi. Updating logical databases. *Advances in Computing Research*, 3:1–18, 1986.

[FLMC02] W. Fan, H. Lu, S. E. Madnick, and D. Cheund. Direct: A system for mining data value conversion rules from disparate data sources. *Decision Support Systems*, 34:19–39, 2002.

[FLPL$^+$01] E. Franconi, A. Laureti Palma, N. Leone, S. Perri, and F. Scarcello. Census data repair: a challenging application of disjunctive logic programming. In *LPAR*, pages 561–578, 2001.

[FM05] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. In *International Conference on Database Theory (ICDT)*, pages 337–351, 2005.

[FR97] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1):32–66, 1997.

[FUV83] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. On the semantics of updates in databases. In *ACM SIGACT-SIGMOD Symposium on Principles* *of Database Systems (PODS)*, pages 352–365. ACM, 1983.

[Gab85] D. Gabbay. Theoretical foundations for non-monotonic reasoning in expert systems. pages 439–457, 1985.

[Gal06] Avigdor Gal. Managing uncertainty in schema matching with top-k schema mappings. *Journal of Data Semantics*, 6:90–114, 2006.

[Gal07] avigdor Gal. Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35(4):2–5, 2007.

[Gar78] Peter. Gardenfors. Conditionals and changes of belief. *Acta Philosophica Fennica*, 30:381–404, 1978.

[Gar88a] P. Gardenfors. The dynamics of belief systems: Foundations vs. coherence. *International journal of Philosophy*, 1988.

[Gar88b] Peter. Gardenfors. *Knowledge in flux : modeling the dynamics of epistemic states*. MIT Press, Cambridge, Mass. :, 1988.

[GD05] L. Getoor and C. P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[GGZ03] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(6):1389–1408, 2003.

[GH92] Dov Gabbay and Anthony Hunter. Making inconsistency respectable 1: A logical framework for inconsistency in reasoning. In *Fundamentals of Artificial Intelligence*, page pages. Springer, 1992.

[GH93] Dov Gabbay and Anthony Hunter. Making inconsistency respectable: Part 2 - meta-level handling of inconsistency. In *In Applied General Systems Research, (ed) G. Klir, Plenum*, pages 129–136. Springer-Verlag, 1993.

[GH06] J. Grant and A. Hunter. Measuring inconsistency in knowledgebases. *Journal of Intelligent Information Systems*, 27(2):159–184, 2006.

[GH08] J. Grant and A. Hunter. Analysing inconsistent first-order knowledge bases. *Artificial Intelligence*, 172(8-9):1064–1093, 2008.

[GIKS03] L. Gravano, P.G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins for data cleansing and integration in an rdbms. In *International Conference on Data Engineering(ICDE)*, pages 729–731, 2003.

[GJ86] J. Grant and Minker. J. Answering queries in indefinite databases and the null value problem. *Advances in Computing Research - The Theory of Databases*, 3:247–267, 1986.

[GL98] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *International Conference on Logic Programming*, pages 1070–1080, 1998.

[GMSS09] Avigdor Gal Gal, Maria Vanina Martinez, Gerardo I. Simari, and V. S. Subrahmanian. Aggregate query answering under uncertain schema mappings. In *International Conference on Data Engineering (ICDE)*, pages 940–951, 2009.

[GN06] Georg Gottlob and Alan Nash. Data exchange: computing cores in polynomial time. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 40–49, New York, NY, USA, 2006. ACM.

[GPSS80] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal basis of fairness. In *Symposium on Principles of Programming Languages (POPL)*, pages 163–173, 1980.

[Gra77] John Grant. Null values in a relational data base. *Inf. Process. Lett.*, 6(5):156–157, 1977.

[Gra78] J. Grant. Classifications for inconsistent theories. *Notre Dame Journal of Formal Logic*, 19(3):435–444, 1978.

[Gra79] John Grant. Partial values in a tabular database model. *Inf. Process. Lett.*, 9(2):97–99, 1979.

[Gra80] John Grant. Incomplete information in a relational database. *Fundamenta Informaticae III*, 3:363–378, 1980.

[Gra91] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer, 1991.

[GS95] John Grant and V. S. Subrahmanian. Reasoning in inconsistent knowledge bases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 7(1):177–189, 1995.

[GS04] Alejandro Javier García and Guillermo Ricardo Simari. Defeasible logic programming: An argumentative approach. *TPLP*, 4(1-2):95–138, 2004.

[Hai84] T. Hailperin. Probability logic. *Notre Dame Journal of Formal Logic*, 25 (3):198–212, 1984.

[Hal90] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, Volume 46(3):pp. 311–350, 1990.

[Han93] Sven Ove Hansson. Reversing the levi identity. *Journal of Philosophical Logic*, 22(6), 1993.

[Han94] Sven Ove Hansson. Kernel contraction. *Journal of Symbolic Logic*, 59(3):845–859, 1994.

[Han97] Sven Ove Hansson. Semi-revision. *Journal of Applied Non-Classical Logic*, (7):151–175, 1997.

[Hec98] D. Heckerman. A tutorial on learning with bayesian networks. *NATO ASI SERIES D BEHAVIOURAL AND SOCIAL SCIENCES*, 89:301–354, 1998.

[HIM$^+$04] A.Y. Halevy, Z.G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza peer data management system. *IEEE Transactions on Knowl-*

edge and Data Engineering (TKDE), 16(7):787–798, 2004.

[HK05] A. Hunter and S. Konieczny. Approaches to measuring inconsistent information. In *Inconsistency Tolerance*, pages 191–236, 2005.

[HL10] Sven Hartmann and Sebastian Link. When data dependencies over sql tables meet the logics of paradox and s-3. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 317–326, New York, NY, USA, 2010. ACM.

[HMYW05] H. He, W. Meng, C. T. Yu, and Z. Wu. Wise-integrator: A system for extracting and integrating complex web search interfaces of the deep web. In *International Conference on Very Large Data Bases (VLDB)*, pages 1314–1317, 2005.

[HS95] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, New York, NY, USA, 1995. ACM.

[IJ81] Tomasz Imielinski and Witold Lipski Jr. On representing incomplete information in a relational data base. In *International Conference on Very Large Data Bases (VLDB)*, pages 388–397, 1981.

[IKBS08] A. Inan, M. Kantarcioglu, E. Bertino, and M. Scannapieco. A hybrid approach to private record linkage. In *International Conference on Data Engineering (ICDE)*, pages 496–505, 2008.

[IL83] T. Imielinski and W. Lipski. Incomplete information and dependencies in relational databases. In *SIGMOD*, pages 178–184, 1983.

[IL84a] Tomasz Imielinski and Witold Lipski. Incomplete information in relational databases. *Journal of ACM*, 31(4):761–791, 1984.

[IL84b] Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[JD05] Wojciech Jamroga and Jürgen Dix. Model checking strategic abilities of agents under incomplete information. In *ICTCS*, pages 295–308, 2005.

[JD06] Wojciech Jamroga and Jürgen Dix. Model checking abilities under incomplete information is indeed delta2-complete. In *EUMAS*, 2006.

[JDR99] P. Jermyn, M. Dixon, and B. J. Read. Preparing clean views of data for data mining. In *ERCIM Workshop on Database Research*, pages 1–15, 1999.

[JKV07] T.S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 346–355, New Orleans, Louisiana, USA, 2007.

[Jr.79] Witold Lipski Jr. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):262–296, 1979.

[KIL04] Gabriele Kern-Isberner and Thomas Lukasiewicz. Combining probabilistic logic programming with the power of maximum entropy. *Artificial Intelligence*, 157(1-2):139–202, 2004.

[KL92] M. Kifer and E. L. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 9(2):179–215, 1992.

[KLM90] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1-2):167–207, 1990.

[KS89] Michael Kifer and V. S. Subrahmanian. On the expressive power of annotated logic programs. In *NACLP*, pages 1069–1089, 1989.

[KS92] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12(3&4):335–367, 1992.

[KTG92] Werner Kiessling, Helmut Thöne, and Ulrich Güntzer. Database support for problematic knowledge. In *International Conference on Extending Database Technology (EDBT)*, pages 421–436, London, UK, 1992. Springer-Verlag.

[LC00] W.-S. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering*, 33(1):49–84, 2000.

[LCcI$^+$04] Chengkai Li, Kevin Chen-chuan, Ihab F. Ilyas, Chang Ihab, F. Ilyas, and Sumin Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD*, pages 131–142. ACM Press, 2004.

[Lev84] Hector J. Levesque. A logic of implicit and explicit belief. In *National Conference on Artificial Intelligence (AAAI)*, pages 198–202, 1984.

[Li09] Xiao Bai Li. A bayesian approach for estimating and replacing missing categorical data. *J. Data and Information Quality*, 1(1):1–11, 2009.

[Lie82] Y. Edmund Lien. On the equivalence of database models. *J. ACM*, 29:333–362, April 1982.

[Lip79] Witold Lipski. On semantic issues connected with incomplete information databases. *ACM Trans. Database Syst.*, 4(3):262–296, 1979.

[Lip81] Witold Lipski. On databases with incomplete information. *Journal of the ACM*, 28(1):41–70, 1981.

[LL98] Mark Levene and George Loizou. Axiomatisation of functional dependencies in incomplete relations. *Theoretical Computer Science*, 206(1-2):283–300, 1998.

[LL99] Mark Levene and George Loizou. Database design for incomplete relations. *ACM Transactions on Database Systems*, 24(1):80–125, 1999.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming, Second Edition*. Springer-Verlag, 1987.

[LLRS97] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.

[Loz94] E. L. Lozinskii. Resolving contradictions: A plausible semantics for inconsistent systems. *Jounal of Automated Reasoning*, 12(1):1–31, 1994.

[LS94] Laks V. S. Lakshmanan and Fereidoon Sadri. Probabilistic deductive databases. In *International Symposium on Logic programming (ILPS)*, pages 254–268, Cambridge, MA, USA, 1994. MIT Press.

[McC87] J. McCarthy. Circumscription—a form of non-monotonic reasoning. pages 145–152, 1987.

[MD80] Drew V. McDermott and Jon Doyle. Non-monotonic logic i. *Artificial Intelligence*, 13(1-2):41–72, 1980.

[ME97] Alvaro Monge and Charles Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records, 1997.

[MHH00] R.J. Miller, L.M. Haas, and M.A. Hernández. Schema mapping as query discovery. In A. El Abbadi, M.L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *International Conference on Very Large Data Bases (VLDB)*, pages 77–88. Morgan Kaufmann, 2000.

[MHH$^+$01] R.J. Miller, M.A. Hernàndez, L.M. Haas, L.-L. Yan, C.T.H. Ho, R. Fagin, and L. Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[MKIS00] E. Mena, V. Kashayap, A. Illarramendi, and A. Sheth. Imprecise answers in distributed environments: Estimation of information loss for multi-ontological based query processing. *International Journal of Cooperative*

Information Systems, 9(4):403–425, 2000.

[MM$^+$11] Maria V. Martinez, Cristian Molinaro, , V.S. Subrahmanian, and Leila Amgoud. A general framework for reasoning about inconsistency. *In preparation*, 2011.

[MMGS11] Maria Vanina Martinez, Cristian Molinaro, John Grant, and V.S. Subrahmanian. Customized policies for handling partial information in relational databases. *Under Review*, 2011.

[Moo85] R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.

[Moo88] R. C. Moore. Autoepistemic Logic. In P. Smets, E. H. Mamdani, D. Dubois, and H. Prade, editors, *Non-Standard Logics for Automated Reasoning*. Academic Press, 1988.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.

[MPP$^+$08] Maria V. Martinez, Francesco Parisi, Andrea Pugliese, Gerardo I. Simari, and V.S. Subrahmanian. Inconsistency management policies. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 367–376, 2008.

[MPP$^+$10] Maria Vanina Martinez, Francesco Parisi, Andrea Pugliese, Gerardo I. Simari, and V. S. Subrahmanian. Efficient policy-based inconsistency management in relational knowledge bases. In *SUM*, pages 264–277, 2010.

[MPS$^+$07] M. V. Martinez, A. Pugliese, G. I. Simari, V. S. Subrahmanian, and H. Prade. How dirty is your relational database? an axiomatic approach. In *ECSQARU*, pages 103–114, 2007.

[MST94] D. Michie, D. J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[Mun74] J. Munkres. *Topology: A First Course*. Prentice Hall, 1974.

[MWJ99] K. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in Artificial Intelligence (UAI)*, page 467475. Citeseer, 1999.

[Nil86a] N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, Volume 28(1):pp. 71–87, 1986.

[Nil86b] Nils Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[NJ02] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 2:841–848, 2002.

[NS92] Raymond Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[NS94] Raymond Ng and V. S. Subrahmanian. Stable semantics for probabilistic deductive databases. *Information and Computation*, 110(1):42–83, 1994.

[OS04] F. Ozcan and V.S. Subrahmanian. Partitioning activities for agents. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 89–113, 2004.

[Pap94] Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[PL92] G. Pinkas and R. P. Loui. Reasoning from inconsistency: a taxonomy of principles for resolving conflicts. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 709–719, 1992.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.

[Poo85] D. Poole. On the comparison of theories: preferring the most specific explanation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 144–147, 1985.

[Poo88] David Poole. A logical framework for default reasoning. *Artificial Intelligence*, Volume 36(1):pp. 27–47, 1988.

[Poo93] David Poole. Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[Poo97] David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

[PS97] Henry Prakken and Giovanni Sartor. Argument-based extended logic programming with defeasible priorities, 1997.

[Pyl99] Dorian Pyle. *Data Preparation for Data Mining (The Morgan Kaufmann Series in Data Management Systems).* Morgan Kaufmann, 1999.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning).* Morgan Kaufmann, 1993.

[RCC92] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 165–176, 1992.

[RDS07] Christopher Ré, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *International Conference on Data Engineering (ICDE)*, pages 886–895, 2007.

[Rei78] R. Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1978.

[Rei80a] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[Rei80b] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[Rei86] Raymond Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33:349–370, April 1986.

[Res64] N. Rescher. Hypothetical reasoning, 1964.

[RM70] N. Rescher and R. Manor. On inference from inconsistent premises. *Theory and decision*, Volume 1:pp. 179–219, 1970.

[Roo92] Nico Roos. A logic for reasoning with inconsistent knowledge. *Artificial Intelligence*, Volume 57(1):pp. 69–103, 1992.

[RSG05] R.B. Ross, V.S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *Journal of the ACM*, 52(1):54–101, 2005.

[SA07] V. S. Subrahmanian and L. Amgoud. A general framework for reasoning about inconsistency. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 599–504, 2007.

[SCM09] Slawomir Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *CoRR*, abs/0908.0464, 2009.

[Sho67] J. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.

[SI07] Mohamed A. Soliman and Ihab F. Ilyas. Top-k query processing in uncertain databases. In *International Conference on Data Engineering (ICDE)*, pages 896–905, 2007.

[SL92] Guillermo R. Simari and Ronald P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2-3):125–157, 1992.

[SNB$^+$08] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[SQL03] Information technology: Database languages, sql part 2 (foundation), 2003.

[SS03] E. Schallehn and K. Sattler. Using similarity-based operations for resolving data-level conflicts. In *BNCOD*, volume 2712, pages 172–189, 2003.

[Tar56] A. Tarski. *On Some Fundamental Concepts of Metamathematics*. Oxford Uni. Press, 1956.

[TK93] K. Thirunarayan and M. Kifer. A theory of nonmonotonic inheritance based on annotated logic. *Artificial Intelligence*, 60(1):23–50, 1993.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[UW02] Jeffrey D. Ullman and Jennifer Widom. *A first course in database systems (2. ed.).* Prentice Hall, 2002.

[Vas79] Yannis Vassiliou. Null values in data base management: A denotational semantics approach. In *SIGMOD*, pages 162–169, 1979.

[Vas80] Yannis Vassiliou. Functional dependencies and incomplete information. In *International Conference on Very Large Data Bases (VLDB)*, pages 260–269, 1980.

[Wij03] J. Wijsen. Condensed representation of database repairs for consistent query answering. In *International Conference on Database Theory (ICDT)*, pages 378–393, 2003.

[Wij05] J. Wijsen. Database repairing using updates. *ACM TODS*, 30(3):722–768, 2005.

[YC88] Li Yan Yuan and Ding-An Chiang. A sound and complete query evaluation algorithm for relational databases with null values. In *SIGMOD*, pages 74–81, New York, NY, USA, 1988. ACM.

[Zan84] C. Zaniolo. Database relations with null values. *Journal of Computer and System Sciences (JCSS)*, 28(1):142–166, 1984.