Personalised Search for the Social Semantic Web

Oana Tifrea-Marciuska

Wolfson College
Department of Computer Science
University of Oxford
Supervised by Prof. Thomas Lukasiewicz
Trinity 2016

A thesis submitted for the degree of
Doctor of Philosophy

Oana Tifrea-Marciuska
Wolfson College

Doctor of Philosophy
Trinity 2016

Personalised Search for the Social Semantic Web

Abstract

Recently, the Web has been changing more and more to what is called the Social Semantic Web. As a consequence, the ranking of search results no longer depends solely on the structure of the interconnections among Web pages.

In my research, I argue that such ranking can be based on user preferences from the Social Web, and on ontological background knowledge from the Semantic Web. Therefore, I combine preference representation languages with Semantic Web technologies.

There is some related research in database community that had dedicated some time to integrate preferences in database queries. However, one cannot directly use the ideas from databases, as we additionally have ontological knowledge, which may introduce unknown values, so-called nulls. Therefore, I need to define the exact semantics and check their feasibility for this context.

In my thesis, as a first step towards closing the gap between the Semantic Web, databases, and preferences, I introduce families of expressive extensions of Datalog$^{\pm}$ with preferences as new paradigms for query answering over ontologies. I first define the syntax and semantic of the proposed frameworks, then propose top-k query answering algorithms under user preferences in semantic data for different types of queries and preference models. Each of the proposed frameworks comes with advantages and disadvantages; therefore, I provide formal properties of my algorithms and empirical experiments on the performance and quality of my results.

Furthermore, I explore the combination of my framework with uncertainty and the generalisation to the preferences of a group of users, where I analyse properties of my algorithms related with social choice theory.

I hereby declare that no part of this thesis has been accepted or is currently being submitted for any degree, diploma or certificate or other qualification in the University of Oxford or elsewhere.

This thesis is submitted to the Department of Computer Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, describes my own research.

Oana Tifrea-Marciuska, Wolfson College

Acknowledgements

'The language of friendship is not words but meanings."

— Henry David Thoreau

In this PhD I propose different languages to be used for the Web. While working on it, I was very lucky to be surrounded by people who toughed me many languages, that gave meaning to my life.

Foremost, I would like to thank my supervisor Prof. Thomas Lukasiewicz, for providing me with the opportunity to come to Oxford. He guided the *language of audacity*: without his encouragement I would have not started this adventure. His guidance, patience and support made this work possible. Thank you for all the letters of recommendation, corrections, discussions, encouragement, patience, knowledge, and all the hard work a supervision involves.

My supervisor kindly introduced me to some great people, with whom I was lucky enough to collaborate. I would like to thank Stefan Borgwardt, Tommaso Di Noia, Bettina Fazzinga, Akanksha Shrivastava, Maria Vanina Martinez, Gerardo I. Simari, David Poole and my supervisor for providing their example to understand the *language of research*: how to approach a problem and how to embrace the unknown. Without their presence and enthusiasm, the PhD would have been a lonely process.

Thank you Giorgio Orsi and Andreas Pieris for their willingness to share their expertise. Giorgio provided me with the Datalog$^{\pm}$ engine and 1-1 tutorials. Andreas dedicated his time to explain to me Datalog$^{\pm}$ classes.

I would like to thank the 50 users that were kind enough to provide me with their preferences for my experiments.

I was lucky to have financial support for half a year from EPSRC and for 3 and half years from Google. Thank you to both institutions for believing in me and my research. Thank you Google for providing me not only with the financial aid, but also with other means to grow as a researcher. I would like to thank to Beate List,

who organised the great Google fellow forums, where I had the chance to present yearly the progress of my research. Having feedback from Google employees helped me hear the industry voice.

I want to thank present and past members of Information systems group and OxWoCS. The great discussions, encouragements and moments spent together made my days more fun. I would like to thank the administration of the Wolfson College, of the Oxford University and of the Department of Computer Science for answering all my emails and being so supportive when needed. They all organised so many interesting events. They all showed me the *language of organisation*.

I would like to thank to my husband Sarunas Marciuska for teaching me the *language of courage* since we met you have encouraged me to try things I thought I cannot do. Once we got the offer from Oxford, he supported the idea to leave Italy and start our UK adventure. Thank you for the love, the unwavering belief in me, for taking care of me when I was sick, being strong when I felt weak, travelling with me, and for his humour when I was taking myself too seriously.

I would like to give a huge thanks to my parents Nicolae Tifrea and Ana Tifrea, who showed me the *language of hard work and persistence*. Thank you for all your love, all the support and the gift of life. You gave me strength to push myself and support when I needed the most. Thank you for taking care of so many things that it would take the whole thesis to enumerate.

I have a deep gratitude for my grandparents Ecaterina Bors and Mihai Bors, who are the masters of the *language of kindness*. They influenced profoundly my character with their faith in the goodness of people and their desire to give selflessly. The phone conversations with my grandmother are invaluable: just few words and I feel fully recharged with her good energy and ready to write few more words on my thesis. Thank you for all your prayers. Even though I lost my grandfather more than 6 years ago, he always lives in my heart. He knew so many funny fables, all of them full of wisdom. Many of those stories helped me during my PhD, and the more

I mature, the more meaning I find in those words.

I would like to thank to my two wonderful sisters Elena Filip and Anca Dobrean. Elena is a very creative person and has found so many ways to show me her unconditional love. She helped me with feedback in presentations and even had sleepless nights because of it. Anca makes me feel like one of the most special persons around her. She always reminds me to listen to my heart and be a fighter. My parents, grandmother, sisters, brothers in law (Teo and Duculet) and four nephews (Alex, Bianca, Tudor and Iustin) make every visit to Romania a moment to await and each add so much to the *language of family*.

I would like to thank to my extended family in Lithuania, my parents in law, Vytutas and Irena Marciuska, and my sister in law Sima Marciuskaite for making me part of the family from the first days we met, showing so many faces of the *language of giving*. Thank you Palione family (Salue, Kristina and Andrius) for all the support and the great time spend together and hosting us in time of need.

I would like to thank to some of my friends Adelina Blaga, Oana Camburu, Maria Nadejde, Evgeny Sherkhonov, Nikola Yolov and Esther Miller for making the friendship world so colourful. Adelina thank you for hosting me so many times, feeding me, for the unforgettable chats and for 18 years of great friendship. You have all enriched my *language of friendship*.

I would like to thank Machine Intelligence and Perception group in Microsoft Research Cambridge for a great internship and introducing me to the language of *industry research*. Special thanks go to Yordan Zaykov for being a great mentor and a role model, and Anna Thomas and Lydia Liu for their friendship.

Special thanks go to my former mentors Irina Datu, Mihai Butika, Gina Natea, Corina Forascu, and Rosella Gennari for encouraging me, helping me grow, and caring for me, making my *language of teaching* so much richer.

I would like to thank my examiners for taking their time to make this thesis better, showing me the importance of the *language of service*.

Contents

List of Figures

List of Tables

Part I

Introduction and Preliminaries

Chapter 1

Introduction

From its inception, the Web has been centred around the idea of linking information to make it more accessible and useful for the users. In the recent years, several important changes have been taking place on the classical Web. First, the so-called Web of Data is increasingly being realised as a special case of the Semantic Web [13]. Second, as a part of the Social Web, users are acting more and more as first-class citizens in the creation and delivery of contents on the Web. The combination of these two technological waves is called the *Social Semantic Web* (or also *Web 3.0*) that contains classical linked information, ontological knowledge and social interactions of users. The *Semantic Web* [13] enables more precise and rich results in search and query answering tasks. Ontology and query languages are examples of Semantic Web technologies used to share, integrate, and query large-scale and less structured data and knowledge bases, such as those occurring on the Web. The *Social Web* contains platforms where people share, view, and search for information (e.g., pictures, texts) in social and sometimes collaborative ways.

In more general level these novelties apply to Big Data in general, where it's hard for end users to satisfy their information needs due to volume, variety, velocity, and complexity dimensions of data. Personalisation and the semantic information is vital in mapping the information needs for users of the Big Data.

With the above mentioned novelties, new scientific challenges arise. First of all, the fast *growth of complex and available data* on the Web makes it hard for end users to satisfy their information needs. Therefore, it is imperative to provide personalised and tailored access that is able to retrieve resources that best fit with the users' interest and preferences. The Social Web can be used to enrich the user profiles, and to pave the way to more sophisticated personalised access to the information, while semantic search has been proposed as an evolution of the keyword-based search that identifies objects, rather than whole documents, as candidates to answering the users' queries.

Moreover, the presence of *uncertainty* in the Web in general is undeniable [67, 89, 113, 132]: information integration (as in travel sites that query multiple sources to find hotels and flights), automatic processing of Web data (analysing an HTML document often involves uncertainty), as well as inherently uncertain data (such as user comments) are all examples of uncertainty that must be dealt with when answering queries over Web.

In addition, modelling the *preferences of a group of users* is also an important research topic in its own right. Many studies have shown that Web search is not a solitary pursuit [58, 130, 159], which motivates the exploration on how this can be realised by providing social inputs. With the growth of social media, people post their preferences and expect to get personalised information. Thus, people use social networks as a tool to organise events, where it is required to combine the individual preferences, and suggest items obtained from aggregated user preferences. As an illustrative example, if there is a movie night of friends, family trip, or dinner with working colleagues, one has to decide which is the ideal movie or location for the group, given the preferences of each member. Another illustrative example is when a company needs to decided products to offer for a promotion. They may receive aggregated preferences of the users preferences from the Social Semantic Web, that can help them optimise their revenue gain.

The aim of my thesis is to make a more formal analysis of the ways that one can do personalised search. To the best of my knowledge, this is something that is definitely missing in this area: a better understanding on how to combine preference representation languages, uncertainty, and the Semantic Web. The Semantic data can provide precise and rich results, while preferences can help us to order the answers of queries. Adding a social side to the Semantic Web requires new techniques for ranking search results, fully exploiting ontological and user-centred data, i.e., user preferences to avoid flooding the users with unnecessary large query results.

Personalisation has been studied in the areas of databases and artificial intelligence. The main difference with this thesis is that besides a database, I have ontological knowledge (i.e., a set of rules and constraints), where one can derive new knowledge from, and that we have to deal with the presence of uncertainty. A query therefore is answered against a database and the ontological knowledge, while the answers are ordered using preferences and uncertainty.

To use preferences for personalisation, one has to deal with the following sub-tasks: preference acquisition, preference modelling, preference representation, and reasoning about preferences [92]. For *preference acquisition*, one can mine preferences by either interacting with the user (preference elicitation) or looking at the past behaviour or preferences of the user (preference learning). In the case of *preference modelling*, one needs to describe the preference relations and their properties (e.g., transitivity). *Preference representation* consists of defining compact encoding from the modelled preferences, where preference relations are not explicitly presented, but implicitly using a compact languages defined to capture and manipulate preferences. For example, when one says that the preferences in a compact language are transitive, one does not need to manually calculate and explicitly encode the transitive closure. Moreover, one may have to define restrictions, to have a clear semantics of the language or nice computational properties. *Preference Reasoning* deals with tasks like query answering, operators for combining preferences, operators for aggregating

the preferences of a group of users.

1.1 Scope of the Thesis

Many ontological languages have been proposed for the Semantic Web; here, this thesis focuses on Datalog$^{\pm}$ that is a language extending Datalog by existentially quantified variables in rule heads. I choose the Datalog$^{\pm}$ language as the ontology language for the Semantic Web, because it is highly flexible (it provides a uniform framework for query answering and reasoning with incomplete data), and it generalises many ontology languages, such as *DL-Lite* family of Description Logics (DLs) [30]. In particular, Datalog$^{\pm}$ has unrestricted predicate arity (DLs consider only unary and binary predicates), which allows for a natural coupling with database schemes, where relations may have any arities, and ideas from the database community related to preferences can be integrated in Datalog$^{\pm}$. Moreover, implementations [78] are also available, and many tractable subclasses of Datalog$^{\pm}$ have been found.

As mentioned above, personalisation requires to do the following sub-tasks: preference acquisition, preference modelling, preference representation, and reasoning about preferences [92]. I focus on the fundamental last three sub-tasks (namely modelling, representations, and reasoning), therefore, preferences acquisition is out of the scope of this thesis. For some experiments, I have acquired preferences by asking the users about their preferences, but the focus is to analyse the other three sub-tasks.

1.2 Research Challenges

Search in the Social Semantic Web may be personalised based on the preferences of a single user or the preferences of a group of users.

The challenges of personalisation consist of defining a suitable language for the Social Semantic Web. Here, we often have to find a compromise between conflicting desired properties, such as: i) having a *natural* language, i.e., to have a language

that sounds naturally; ii) having an *expressive* language, i.e., can handle diverse user preferences, and can effectively convey preferences; iii) having a *clear semantics* for the language, i.e., represent the users' preferences truly in the sense that it orders choices according to user's preferences over them; iv) coping with the presence of *inconsistency*; v) *compactness*, so users do not spend too much time to express all their preferences (cognitive limitation); vi) *efficiency* in terms of space complexity (i.e., cost to represent preferences) and time complexity (i.e., cost to do the reasoning tasks); vii) coping with *bipolar preferences* (what a user really wants and what he does not want in his answers); viii) leveraging the *social components* (preferences of a user or a group of users) of the Web content towards the development of some form of semantic search and query answering on the Web; ix) defining a *group preference* semantics that solves disagreements among users—for instance, people (even friends) often have different tastes in restaurants; a system should return results in such a way that certain properties are satisfied e.g., ensuring that each individual benefits from the result.

There are a number of preference representation languages and frameworks in AI and, there are many query languages in the Semantic Web already. However, the challenge remains to understand which of these ideas could be extended towards Semantic Web personalisation.

Example 1.2.1. Consider a simple situation from daily life in which three friends, Alberto, Bruno, and Charles, decide where to go on holidays, choosing from three possible locations: $Prague$, $London$, and $Paris$. Alberto's preferences are $\langle Prague, \{London, Paris\}\rangle$ (i.e., he prefers $Prague$, and then he does not have any preference between $London$ and $Paris$), Bruno's are $\langle London, \{Prague, Paris\}\rangle$, and Charles's are $\langle Prague, London, Paris\rangle$. However, the three friends have an additional information from a review site that gives a score to each location stating how likely it is that it will be too crowded; these probabilities are 0.2 for $Prague$, 0.9 for $London$, and 0.6 for $Paris$.

Taking a simple voting approach only from the preferences of the users (no uncertainty) where the majority decides would lead the friends to choose *Prague*, since it is the first choice of two of them; however, it is also the least likely to have good accommodation deals available — *London*, on the other hand, seems to be a better choice, since it is well-ranked by both Bruno and Charles, and has a high probability as well. ∎

1.3 Research Goals

By addressing the enumerated challenges I add a novel dimension to the fundamental problem of search in the Web, aiming at retrieving precise and rich results. I analyse the best combination (natural, expressive, clear, compact, efficient) of Semantic Web languages, preference representation languages, and uncertainty formalisms to answer personalised queries for a single user or a group of users. Combining them requires understanding the advantages and disadvantages of each of these languages.

The goal of my thesis is threefold:

- First, it aims at bridging a gap between Semantic Web languages and preference representation languages. To address the following, it defines the syntax and semantic of the three novel preference frameworks, and describe how to compute top-k answers under these frameworks.

- Second, it aims to understand the advantages and disadvantages of each of these languages (natural, expressive, clear, compact, efficient). Therefore, it defines formal properties of my algorithms, and perform empirical experiments on the performance and quality of these algorithms.

- Last, but not least, it generalises the usage of my preference framework to group of users, and to the presence of uncertainty.

1.4 Summary of the Contributions

We analyse different preference representations: i) qualitative (e.g., "I prefer London to Paris") versus quantitative (e.g., "I like London 0.7"); ii) conditional (e.g., "If it rains, I prefer Paris to London") versus independent (e.g., "I prefer London to Paris, independently of the weather"); iii) variable priority (e.g., "the price is more important than location when choosing a hotel") or not; iv) single user versus group of users.

The results in this work can be summarised as follows: single-user qualitative preferences, group-of users qualitative preferences, single user conditional preferences, single user quantitative preferences, group of user quantitative preferences. This thesis includes and extends results published in the following papers: [50, 117–121, 133, 134, 157].

1.4.1 Single-User Qualitative Preferences

Here I address the following two challenges: i) handling diverse user preferences, i.e., to have a natural language that is very expressive and, ii) coping with the presence of inconsistency and uncertainty. I developed an extension of Datalog$^{\pm}$ with two models: one representing partial qualitative preferences (e.g., "I prefer location *London* to *Paris*"), and one representing uncertainty (e.g., "location *London* has probability 0.9 of being crowded"), called PP_Datalog$^{\pm}$. This section includes and extends results published in [121].

The main contributions of this section can be briefly summarised as follows.

- It introduces the PP_Datalog$^{\pm}$ framework, which combines ontology languages with both preferences as in relational databases, and probabilistic uncertainty.
- It formalises the concept of *preference combination* operators, which take as input a preference relation in the form of a strict partial order (SPO) and a score-based SPO (a weak order), and produce a new preference relation

satisfying certain basic properties.

- It develops four specific preference combination algorithms, two egalitarian ones (ComPrefsGen and ComPrefsRank) and two user-biased ones (ComPrefsPT and ComPrefsSort). They satisfy the requirements of a preference combination operator as well as other desirable properties, among which are several postulates from the literature for a less general case. They can also be implemented to run in polynomial time in the data complexity modulo the cost of computing probabilities.

- It presents an algorithm for answering k-rank queries, a generalisation of top-k queries based on the iterative computation of classical skyline answers, for disjunctions of atomic queries (DAQs), along with proofs of correctness and running time, showing that answering DAQs in PP_Datalog$^{\pm}$ is possible in polynomial time in the data complexity modulo the cost of computing probabilities.

- It illustrates PP_Datalog$^{\pm}$ along a real-world application with preference-based query answering in the context of the Internet Movie Database (IMDB).

- It also reports on an implementation of PP_Datalog$^{\pm}$ along with extensive experimental results, evaluating and analysing the running time of the algorithms over a combination of real-world and synthetic data.

1.4.2 Group-of-Users Qualitative Preferences

Here I address the following challenges: i) handling diverse user preferences, i.e., to have a natural language that is very expressive; ii) coping with the presence of inconsistency and uncertainty; iii) handling preferences of a group of users. I propose a novel integration of Datalog$^{\pm}$ that both allows for the management of partially ordered preferences of groups of users and as well as uncertainty, which is represented via a probabilistic model. The language, named GPP_Datalog$^{\pm}$, is the

extension of PP_Datalog$^{\pm}$ to a group of users. This section includes and extends results published in [118–120].

The main contributions of this section can be summarised as follows.

- It introduces GPP_Datalog$^{\pm}$, a language that combines the Datalog$^{\pm}$ [31] ontology language with group preferences (a generalisation of preference handling in relational databases) and probabilistic uncertainty. To my knowledge, this is the first combination of ontology languages with group preferences (with and without probabilistic uncertainty). The preference and the probabilistic models are assumed to represent the preferences of a group of users, and the uncertainty in the domain, respectively.

- It formalises the notion of k-rank query answering based on operators for merging single-user and probability-based preferences (in the form of a strict partial and a weak order, respectively), and aggregating multiple single-user preference relations.

- It analyses two approaches to computing an answer to a k-rank query that are suitable for partially ordered sets of preferences: collapse to single user (CSU), which constructs a single virtual user that aggregates the preferences of all the individuals from the group, and the k-rank answers are computed over this new preference relation; and voting-based aggregation, where k-rankings are computed first for each individual user and then aggregation techniques based on voting strategies are used to aggregate the answers and obtain a single k-ranking.

- Based on an algorithm for the above preference merging and aggregation, it gives algorithms for answering k-rank queries for DAQs (disjunctions of atomic queries), which generalise top-k queries based on the iterative computation of classical skyline answers. I show that answering DAQs in GPP_Datalog$^{\pm}$

is possible in polynomial time in the data complexity modulo the cost of computing probabilities.

- Finally, it develops a prototype implementation of a group-preference-based query answering system, and conducted a series of experiments based on real-world ontological data and preference models derived from information gathered from real users. The results (on the running time of my algorithms and the quality of their results) show in particular that the strategies proposed and developed in this work are feasible in practice.

1.4.3 Single User Conditional Preferences

Here I address the following challenges: i) representing the user preferences truly in the sense that it orders choices with the user's specification over them; ii) compactness, such that the user does not spend too much time to express all their preferences (cognitive limitation); iii) efficiency in terms of space complexity (i.e., cost to represent preferences) and time complexity (i.e., cost to do reasoning tasks). More specifically, I introduce ontological CP-theories. The need for this formalism arises, because answering conjunctive queries in PP_Datalog$^{\pm}$ was proven intractable. This section includes and extends results published in [50, 133, 134].

The main contributions of this section are briefly as follows:

- It introduces ontological CP-theories (OCP-theories), which combine Datalog$^{\pm}$ with CP-theories, modelling preferences over ground atoms in Datalog$^{\pm}$ ontologies.

- It defines skyline and k-rank answers for conjunctive queries (CQs) to OCP-theories. I also provide an algorithm for computing such answers based on the preferences encoded in an OCP-theory.

- It analyses the computational complexity of deciding consistency, dominance, and CQ skyline membership for OCP-theories, providing (generic and concrete)

precise complexity results for different types of combined complexity.

- It also provides several tractability results in the data complexity for the case where query answering in the underlying classical ontology is tractable in the data complexity.

1.4.4 Single User Quantitative Preferences

In this part I address the following challenges: i) efficient in terms of space complexity (i.e., cost to represent the language) and time complexity (i.e., cost to do the reasoning tasks); ii) cope with bipolar preferences (what a user really wants and what he does not want in his answers); iii) a formalism that is based on scores on atoms.

The intractability of evaluating CQs of the PP_Datalog$^{\pm}$ motivated the search for tractable special cases. Note that the algorithms of the other formalisms are not optimised for score-based preferences, as they do not leverage this simpler structure.

The main contributions of this section are briefly as follows:

- It proposes a language, named SP_Datalog$^{\pm}$, that is an extension of Datalog$^{\pm}$ with score based preferences. We introduce an approach to top-k query answering under SP_Datalog$^{\pm}$, where the queries are unions of conjunctive queries with safe negation.

- It develops an algorithm for top-k query answering in this framework, which is based on a generalisation of the previous RankJoin algorithm [85] to ontology-based data access and to union of conjunctive queries with safe negation as queries.

- It proves the correctness of the proposed algorithm and provide a running example.

- It proves that the special case of score-based preferences allows us to compute UCQ answers in polynomial time.

1.4.5 Group of Users Quantitative Preferences

Here I address the following challenges: i) efficiency in terms of space complexity (i.e., cost to represent the language) and time complexity (i.e., cost to do the reasoning tasks); ii) coping with bipolar preferences (what a user really wants and what he does not want in his answers) iii) handling preferences of a group of users. I extend the previous formalism to a group of users.

The main contributions of this work are briefly as follows:

- It proposes a new language GSP_Datalog$^{\pm}$, that generalises SP_Datalog$^{\pm}$ to a group of users, rather than a single user.
- It provides an algorithm for answering top-k unions of conjunctive queries with safe negation for GSP_Datalog$^{\pm}$ language, which involves the aggregation of (potentially conflicting) user preferences.
- It studies two different approaches to do aggregations of users preferences and analyse their properties.
- It provides experimental results on the performance (in terms of running time) and the quality of my algorithms (using standard measures from information retrieval. It explores which of the techniques for aggregating user preferences is the best and how much their results differ).

1.5 Structure of this Thesis

The remainder of the thesis is organised as follows. In Chapter 2, I present preliminary definitions and necessary background material. We recall some basics on Datalog$^{\pm}$, conjunctive queries, the query answering problem for Datalog$^{\pm}$, review and survey various Datalog$^{\pm}$ sub-languages that provide decidability guarantees for Boolean conjunctive query answering, and the preferences representation languages. In Chapter 3, I introduce a language that can express qualitative preferences, and is

able to handle uncertainty. I analyse how to combine uncertainty with preferences to obtain ordered answers when one does query answering. Chapter 4 extends the work of the previous chapter to preferences for a group of users. I define aggregation operators of the users' preferences from the group, and define query answering for this language. In Chapter 5, I define a language that combines conditional preferences, and define exact complexity results for different fragments of this language for top-k query answering. In Chapter 6, I define an ontological score-based preference language for a single user, while in Chapter 7, I extend it for a group of users. In both chapters, I present top-k query answering.

Chapter 8 reviews the related literature, covering especially prior work on preference modelling. Finally, Chapter 9 concludes with a discussion of the obtained results, as well as an overview of directions for future research and some open problems.

Chapter 2

Preliminaries

2.1 Datalog$^{\pm}$

This section introduces the necessary preliminaries to present the novel frameworks: Datalog$^{\pm}$ and preference representation languages. For Datalog$^{\pm}$, I present its syntax, semantics, its query answering engine and complexity results for query answering. In case of preference representation languages, I present qualitative preferences, CP-theories, and PrefDatalog$^{\pm}$.

2.1.1 Datalog$^{\pm}$ Preliminaries

Datalog$^{\pm}$ [31] is a rule-based formalism that combines the advantages of logic programming in Datalog with features for expressing ontological knowledge. It is a formalism that extends plain Datalog with features such as existential quantifiers, equalities, and the falsum ($\bot$) in rule heads and, at the same time, restricts the rule syntax so as to achieve decidability and, when required, tractability.

Relational Databases. Let us consider (i) an infinite universe of *(data) constants* Δ (which constitute the "normal" domain of a database), (ii) an infinite set of *(labelled) nulls* Δ_N (used as "fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables $\mathcal{V}$ (used in queries, dependencies, and constraints). Different constants

represent different values (*unique name assumption*), while different nulls may represent the same value. Let us assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. Let us denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geq 0$.

Let us assume a *relational schema* $\mathcal{R}$, which is a finite set of *predicate symbols* (or simply *predicates*). A *term* t is a constant, null, or variable. An *atomic formula* (or *atom*) $\mathbf{a}$ has the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. A term or atom is *ground* if it contains no nulls and no variables. An *instance* I for a relational schema $\mathcal{R}$ is a (possibly infinite) set of atoms with predicates from $\mathcal{R}$ and arguments from $\Delta \cup \Delta_N$. A *database* is a finite instance that contains only constants (i.e., its arguments are from Δ).

Let us define the notion of a homomorphism h, from a set of atoms A_1 to a set of atoms A_2, that is a mapping $h\colon \Delta \cup \Delta_N \cup \mathcal{V} \rightarrow \Delta \cup \Delta_N \cup \mathcal{V}$ such that (i) $c \in \Delta$ implies $h(c) = c$, (ii) $c \in \Delta_N$ implies $h(c) \in \Delta \cup \Delta_N$, and (iii) $r(t_1, \ldots, t_n) \in A_1$ implies $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))) \in A_2$. Similarly, one can extend h for a conjunction of atoms.

2.1.1.1 Syntax and Semantics of Datalog$^{\pm}$

Given a relational schema $\mathcal{R}$, a Datalog$^{\pm}$ program consists of a finite set of tuple-generating dependencies (TGDs), negative constraints (NCs) and equality-generating dependencies (EGCs).

1. A *tuple-generating dependency (TGD)* σ is a first-order (FO) rule that allows existential quantified conjunctions of atoms in rule heads:

$$\sigma \ : \forall \mathbf{X} \forall \mathbf{Y} \ \underbrace{\Phi(\mathbf{X}, \mathbf{Y})}_{\text{body}(\sigma)} \rightarrow \underbrace{\exists \mathbf{Z} \ \Psi(\mathbf{Y}, \mathbf{Z})}_{\text{head}(\sigma)} \text{ with } \mathbf{X}, \mathbf{Y}, \mathbf{Z} \subset \mathcal{V},$$

   where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms. Note that TGDs with multiple single atoms in the head can be converted into sets of TGDs with only single atoms in the head [29]. Therefore, in the sequel, we assume that all sets

of TGDs have only single atoms in their head.

An instance I for $\mathcal{R}$ *satisfies* σ, denoted $I \models \sigma$, if whenever there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of I, there exists an extension h' of h that maps $\Psi(\mathbf{X}, \mathbf{Z})$ to atoms of I (i.e, $h' \supseteq h, h'(\Psi(\mathbf{X}, \mathbf{Z})) \in I$).

2. A *negative constraint (NC)* ν is an FO rule that allows to express negation:

$$\nu \ : \ \underbrace{\forall \mathbf{X} \ \Phi(\mathbf{X})}_{\text{body}(\nu)} \rightarrow \bot \qquad\qquad \text{with } \mathbf{X} \subset \mathcal{V},$$

where $\Phi(\mathbf{X})$ a conjunction of atoms. An instance I for $\mathcal{R}$ *satisfies* ν, denoted $I \models \nu$, if for each homomorphism h, $h(\Phi(\mathbf{X}, \mathbf{Y})) \not\subseteq I$ holds.

3. An *equality-generating dependency (EGD* μ is an FO rule

$$\mu \ : \ \underbrace{\forall \mathbf{X} \, \Phi(\mathbf{X})}_{\text{body}(\mu)} \rightarrow X_i = X_j \qquad\qquad \text{with } X_i, X_j \in \mathbf{X} \subset \mathcal{V},$$

where $\Phi(\mathbf{X})$ is conjunction of atoms. An instance I for $\mathcal{R}$ *satisfies* μ, denoted $I \models \mu$, if whenever there is a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq I$, it holds that $h(X_i) = h(X_j)$.

A *Datalog*$^{\pm}$ *program* Σ is a finite set $\Sigma_T \cup \Sigma_{NC} \cup \Sigma_E$ of TGDs, NCs and EGDs. The schema of Σ , denoted $\mathcal{R}(\Sigma)$, is the set of predicates occurring in Σ. A *Datalog*$^{\pm}$ *ontology* $O = (D, \Sigma)$ consists of a finite database D and a program Σ.

The conjunction of the first-order sentences associated with the rules of a Datalog$^{\pm}$ program Σ is denoted Σ_P. A *model* of Σ is an instance for $\mathcal{R}(\Sigma)$ that satisfies Σ_p. For a database D for $\mathcal{R}$, and a set of TGDs Σ on $\mathcal{R}$, the set of *models* of D and Σ, denoted $mods(D, \Sigma)$, is the set of all (possibly infinite) instances I such that (i) $I \subseteq B$ and (ii) every $\sigma \in \Sigma$ is satisfied in I (i.e., $I \models \Sigma$). The ontology is *consistent* if the set $mods(D, \Sigma)$ is not empty.

The semantics of Σ on an input database D, denoted $\Sigma(D)$, is an instance I for

$\mathcal{R}(\Sigma)$ such that $I \supseteq D$, I is a model of Σ, and for each model I' of Σ containing D, there exists a homomorphism h such that $h(I) \subseteq I'$; such an instance is called *universal model* of Σ w.r.t. D. Intuitively speaking, a universal model contains no more and no less information than what the given program requires.

In general, there exists more than one universal models of Σ w.r.t. D, but the universal models are (by definition) the same up to homomorphic equivalence, i.e., for each pair of universal models M_1 and M_2, there exist homomorphisms h_1 and h_2 such that $h_1(M_1) \subseteq M_2$ and $h_2(M_2) \subseteq M_1$. Thus, $\Sigma(D)$ is unique up to homomorphic equivalence.

The following example shows a simple Datalog$^{\pm}$ program.

Example 2.1.1. A Datalog$^{\pm}$ ontology $O = (D, \Sigma)$ for the real estate domain is given below with the database D in Table 2.1.

Intuitively, D encodes that p_1, p_2, p_3, p_4, p_5, and p_6 are properties, and with the exception of p_4, which is a house, all the other properties are apartments. The TGDs in Σ encode, for example, the domain of the feature property and of the offers. It shows as well that the friend relation is not reflexive, and that there are no properties with same id with two different types. Moreover it encodes that for a the $\mathit{property}(P, T, L, C)$, the variable P stands for a *place*. Let us denote by $\mathbf{t}_1$ the term $\mathit{property}(p_1, \mathit{apt}, l_1, e)$ and by t_1 the tuple $(p_1, \mathit{apt}, l_1, e)$.

$$
\begin{aligned}
\Sigma = \{ & \mathit{propertyFeature}(P, F) \rightarrow \exists T, L, C \ \mathit{property}(P, T, L, C), \\
& \mathit{offer}(U, P) \rightarrow \exists T, L, C \ \mathit{property}(P, T, L, C), \\
& \mathit{offer}(U, P) \rightarrow \exists R \ \mathit{user}(U, R), \ \mathit{property}(P, T, L, C) \rightarrow \mathit{location}(L), \\
& \mathit{property}(P, T_1, L_1, C_1) \wedge \mathit{property}(P, T_2, L_2, C_2) \rightarrow T_1 = T_2, L_1 = L_2, C_1 = C_2 \\
& \mathit{friend}(P, P) \rightarrow \bot, \ \mathit{property}(P, T, L, C) \rightarrow \mathit{place}(P)\}.
\end{aligned}
$$

■

Table 2.1: Database D

	id	*type*	*location*	*class*
t_1	p_1	apt	l_1	e
t_2	p_2	house	l_2	l
t_3	p_3	apt	l_1	e
t_4	p_4	apt	l_2	l
t_5	p_5	apt	l_2	e
t_6	p_6	apt	l_3	e

property

	property	*feature*
t_{10}	p_1	g
t_{11}	p_2	p
t_{12}	p_3	p
t_{13}	p_3	g

propertyFeature

	user	*property*
t_{14}	b	p_2
t_{15}	b	p_1
t_{16}	j	p_4
t_{17}	j	p_5

offer

	user	*user*
t_{18}	b	a
t_{19}	j	a
t_{20}	b	m

friend

	id	*review*
t_7	b	p
t_8	j	n
t_9	a	p

user

2.1.1.2 Conjunctive Query Answering

Let us now introduce conjunctive query answering for Datalog$^{\pm}$. A *conjunctive query (CQ)* over $\mathcal{R}$ has the form:

$$\underbrace{q(\mathbf{X})}_{\text{head}} \ :- \ \underbrace{\exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})}_{\text{body}}, \tag{2.1}$$

where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms (possibly equalities, but not inequalities) with the variables $\mathbf{X}$ and $\mathbf{Y}$, and possibly constants, but without nulls, and q is a predicate not occurring in $\mathcal{R}$. A *Boolean CQ (BCQ)* over $\mathcal{R}$ is a CQ of the form $q()$, often written as the set of all its atoms, without quantifiers.

The set of all *answers* to a CQ $q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$ over an instance of a database I, denoted $q(I)$, is the set of all tuples $\mathbf{t}$ over Δ, for which there exists a homomorphism $h\colon \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ $q()$ over a database instance I is *Yes*, denoted $D \models q$, if $q(I) \neq \emptyset$. Formally, *query answering* under TGDs, i.e., the evaluation of CQs and BCQs on databases under a set of TGDs is defined as follows. The set of *answers* for a CQ q to D and Σ, denoted

$ans(q, D, \Sigma)$, is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in q(I)$ for all $I \in mods(D, \Sigma)$. The *answer* for a BCQ q to D and Σ is *Yes*, denoted $D \cup \Sigma \models q$, if $ans(q, D, \Sigma) \neq \emptyset$. Note that for query answering, homomorphically equivalent instances are indistinguishable, i.e., given two instances I and I' that are the same up to homomorphic equivalence, $q(I)$ and $q(I')$ coincide. Therefore, queries can be evaluated on any universal model.

The decision problem of *CQ answering* is defined as follows: given a database D, a set Σ of TGDs and NCs, a CQ q, and a tuple of constants $\mathbf{t}$, decide whether $\mathbf{t} \in ans(q, D, \Sigma)$.

For query answering of BCQs in Datalog$^{\pm}$ with TGDs, adding negative constraints is computationally easy, as for each constraint $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \bot$ one only has to check that the BCQ $\exists \mathbf{X}\, \Phi(\mathbf{X})$ evaluates to false in D under Σ; if one of these checks fails, then the answer to the original BCQ q is true, otherwise the constraints can simply be ignored when answering the BCQ q.

Adding EGDs over databases with TGDs along with negative constraints does not increase the complexity of BCQ query answering as long as they are *non-conflicting* [31]. Intuitively, this ensures that, if the chase (see below) fails (due to strong violations of EGDs), then it already fails on the database, and if it does not fail, then whenever "new" atoms are created in the chase by the application of the EGD chase rule, atoms that are logically equivalent to the new ones are guaranteed to be generated also in the absence of the EGDs, guaranteeing that EGDs do not influence the chase with respect to query answering. Therefore, for the rest of this paper we assume that all the fragments of Datalog$^{\pm}$ have non-conflicting rules.

2.1.1.3 The TGD Chase

Query answering under general TGDs is undecidable [9] and the chase is used as a procedure to do query answering for Datalog$^{\pm}$. Given a Datalog$^{\pm}$ program Σ with only TGDs (see [31] for further details and for an extended chase with also EGDs), $\Sigma(D)$ can be defined as the least fixpoint of a monotonic operator (modulo homomorphic equivalence). This can be achieved by exploiting the well-known *chase*

procedure, originally introduced for checking implication of dependencies, and for checking query containment [79]. The chase is an algorithm that, roughly speaking, executes the rules of Σ starting from D in a forward chaining manner by inferring new atoms, and inventing new null values whenever an existential quantifier needs to be satisfied. By "chase", I refer both to the chase procedure and to its output.

Let D be a database, and σ a TGD of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{Y}, \mathbf{Z})$. Then, σ is *applicable* to D, if there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let σ be applicable to D, and h_1 be a homomorphism that extends h as follows: for each $Y_i \in \mathbf{Y}$, $h_1(Y_i) = h(Y_i)$; for each $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N$, z_j does not occur in D, and z_j lexicographically follows all other nulls already introduced. The *application of* σ on D adds to D the atom $h_1(\Psi(\mathbf{Y}, \mathbf{Z}))$ if not already in D.

The chase algorithm for a database D and a set of TGDs Σ consists of an exhaustive application of the TGD chase rule in a breadth-first (level-saturating) fashion, which outputs a (possibly infinite) chase for D and Σ.

Formally, the *chase of level up to* 0 of D relative to Σ, denoted $chase^0(D, \Sigma)$, is defined as D, assigning to every atom in D the *(derivation) level* 0.

For every $k \geq 1$, the *chase of level up to* k of D relative to Σ, denoted $chase^k(D, \Sigma)$, is constructed as follows: let $I_1, \ldots, I_n$ be all possible images of bodies of TGDs in Σ relative to some homomorphism such that (i) $I_1, \ldots, I_n \subseteq chase^{k-1}(D, \Sigma)$ and (ii) the highest level of an atom in every I_i is $k-1$; then, perform every corresponding TGD application on $chase^{k-1}(D, \Sigma)$, choosing the applied TGDs and homomorphisms in a (fixed) linear and lexicographic order, respectively, and assigning to every new atom the *(derivation) level* k. The *chase* of D relative to Σ, denoted $chase(D, \Sigma)$, is defined as the limit of $chase^k(D, \Sigma)$ for $k \rightarrow \infty$.

The (possibly infinite) chase of a database D relative to a set of TGDs Σ, denoted $chase(D, \Sigma)$, is a *universal model*, i.e., there is a homomorphism from $chase(D, \Sigma)$ onto every $B \in mods(D, \Sigma)$ [31]. Thus, BCQs q over D and Σ can be evaluated on

the chase for D and Σ, i.e., $D \cup \Sigma \models q$ is equivalent to $chase(D, \Sigma) \models q$.

Example 2.1.2. Consider a database $D = \{person(john)\}$, and the ontological theory Σ consisting of the constraints $\Sigma = \{person(X) \rightarrow \exists Y father(Y, X)$, $father(X, Y) \rightarrow person(X)\}$, stating that every person has a father, who is himself a person. Then, $chase(D, \Sigma)$ is the infinite set of atoms $D \cup \{father(z_1,\ john)$, $person(z_1),\ father(z_1,\ z_2),\ person(z_2), father(z_2, z_3), person(z_3), ...\}$, where each z_i are labelled null values [34]. Even for the Σ consisting only of $r(X, Y) \rightarrow \exists Z r(Y, Z)$ would lead an infinite chase for $D = \{r(a, b)\}$. ∎

There are two ways of processing rules: forward chaining (the chase), introduced above, and backward chaining. The backward chaining uses the rules to rewrite the query in different ways with the aim of producing a query that maps to the facts. The key operation is the unification between part of a current goal (a conjunctive query or a fact in my framework) and a rule head. The chase can be seen as forward chaining operation. Let us assume that the nulls introduced in the chase are named via Skolemization (that also makes the chase unique), and so that Δ_N is the set of all nulls that may be thus introduced in the chase.

2.1.2 Complexity Measures of Datalog$^{\pm}$

Following Vardi's taxonomy [162], the *combined complexity* of CQ answering is calculated by considering all the components, i.e., the database, the set of dependencies, and the query, as part of the input.

The *bounded-arity combined complexity* (or *ba-combined complexity*) is calculated by assuming that the arity of the underlying schema is bounded by an integer constant. Notice that in the context of DLs, whenever we refer to the combined complexity in fact we refer to the *ba*-combined complexity, as the arity of the underlying schema is at most two. In practical applications, the schema is usually small, and it could be assumed to be fixed.

The *fixed-program combined complexity* (or *fp-combined complexity*) is calculated

by considering the set of dependencies as fixed.

The *data complexity* of CQ answering is calculated by taking only the database as input.

2.1.3 General Complexity Results

Query answering under general TGDs is undecidable [9], even when the query q and Σ are fixed [29]. The two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent [28]. Moreover, the query output tuple (QOT) problem (as a decision version of CQ evaluation) and BCQ evaluation are AC_0-reducible to each other. Henceforth, I thus focus only on BCQ evaluation, and any complexity results carry over to the other problems.

2.1.4 Decidability Paradigms

In this section, I discuss different restrictions that ensure the decidability of answering CQs for TGDs.

Generally, these restrictions can be classified into either *abstract* (or semantic) or *concrete* (or syntactic) properties. The abstract properties focus on the way the reasoning algorithm behaves. Three abstract properties related to the behaviour of reasoning mechanisms are considered in [7]: the chase is finite; the chase may not halt but the facts generated have a tree-like structure; the backward chaining mechanism halts in finite time. These properties yield three abstract classes of rules, called finite expansion sets (fes), bounded tree-width sets (bts), and finite unification sets (fus), respectively. Recently, a new abstract fragment has been identified as parsimonious sets (ps) [105], where TGDs under which the chase can be precociously terminated. Another abstract class [128] weakly-chase-sticky that applies only to values for repeated variables in the body of the rules that appear in so-called infinite positions (if there is an instance D, for which an unlimited number of different values appear in its chase), which are semantically defined.

The main (syntactic) conditions on TGDs that guarantee the decidability of CQ

answering are guardedness [28, 30], stickiness [35], and acyclicity. Interestingly, each of them has its "weak" counterpart: weak guardedness [29], weak stickiness [35], and weak acyclicity [60, 61], respectively.

2.1.4.1 Classes of TGDs based on Guardedness

This section introduces various classes of TGDs that are based on guardedness.

Guarded TGDs. Informally, a TGD is guarded if it has a body atom that contains (or "guards") all body variables.

Definition 2.1.1. A TGD σ is *guarded*, if it is of the form $\sigma : \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \Psi(\mathbf{Y}, \mathbf{Z})$, with $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subseteq \mathcal{V}$, $\mathbf{X} \cap \mathbf{Z} = \emptyset$, where, $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Phi(\mathbf{X}, \mathbf{Y})$ are conjunctions of atoms, and $r(\mathbf{X}, \mathbf{Y})$ is an atom of $\Phi(\mathbf{X}, \mathbf{Y})$. The atom $r(\mathbf{X}, \mathbf{Y})$ is the guard of σ.

Example 2.1.3. The TGD $\sigma_1 : r(A, B, C) \wedge p(B, C) \rightarrow \exists Y\, r(A, B, Y)$ is guarded with $r(A, B, C)$, the guard of σ_1. The TGD $\sigma_2 : r(A, B, C) \wedge p(B, D) \rightarrow \exists Y\, r(A, B, Y)$ is not guarded, since there is no atom that contains all the variables A, B, C, and D.

The class of guarded TGDs, denoted G, is defined as the family of all possible sets of guarded TGDs.

Linear TGDs. A key subclass of guarded TGDs are linear TGDs, which have just one body atom (which is automatically a guard), and the corresponding class is denoted L. The class allows for self-joins and variable repetition, both in the head and the body of a TGD.

Definition 2.1.2. A TGD σ is *linear*, if it is of the form $\sigma : r(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \Psi(\mathbf{Y}, \mathbf{Z})$, with $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subseteq \mathcal{V}$ are pairwise disjoint, $\Psi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms, and r is a relation.

Example 2.1.4. The TGDs $\sigma_3 : p(X, X) \rightarrow \exists Y\, \mathit{reflexive}(X) \wedge s(X, Y)$ is linear. The TGDs σ_1 and σ_2 from Example 2.1.3 are not linear.

Weakly guarded TGDs. Informally, a weakly guarded set of TGDs extend sets of guarded TGDs by requiring only "harmful" body variables to appear in the guard, i.e., weakly guardedness requires guards to cover all variables occurring in affected positions only, where affected positions are positions in predicates that may contain some fresh labelled nulls generated during the construction of the chase [29]. The associated class is denoted WG. Notice that $\mathsf{L} \subset \mathsf{G} \subset \mathsf{WG}$.

Definition 2.1.3. Given a set of TGDs σ over a relational schema R, the set of *affected positions*, is defined inductively as the least fixpoint of the following:

- Let $r[i]$ be a position in the head of a TGD $\sigma \in \Sigma$. If an existentially quantified variable appears at position $r[i]$ in σ, the $r[i]$ is affected.
- Let $r[i]$ be a position in the head of a TGD $\sigma \in \Sigma$. If a universally quantified variable appears at position $r[i]$, that variable only appears at affected positions in the body of σ, then $r[i]$ is affected.

Definition 2.1.4. A set Σ of TGDs is *weakly guarded* if for each TGD $\sigma \in \Sigma$, it holds that all universally quantified variables that appear only at affected positions in the body of σ appear together in a body atom. This atom is the weak guard of σ.

Example 2.1.5. Let $\Sigma = \{\sigma_1, \sigma_3, \sigma_4\}$, with σ_1 and σ_3 from Examples 2.1.3 and 2.1.4, and $\sigma_4 : s(X, Y) \wedge s(Y, Z) \rightarrow s(X, Z)$. The Σ is weakly guarded, with the weak guard of σ_4 being $s(Y, Z)$, since the affected position of σ_4 is $s[2]$. If we add $\sigma_5 : r(X, Y, X) \rightarrow \exists Z\ s(Z, X)$ to Σ, then Σ is not a weak guard, since both positions of the relation s are affected, and σ_4 does not contain an atom with all the X, Y, Z together.

Note that the weakly-guardedness property cannot be checked at the level of each TGD (as it can be done for the L and G classes), but rather one needs to take all the TGDs into account, to determine the affected positions.

Frontier-Guarded TGDs. Frontier guardedness (denoted YG) [5] relaxes the guardedness conditions of TGDs by requiring the guard atom to contain only the frontier of the TGD. The *frontier* of a TGD is the set of all variables that appear in both the body and the head of a variable.

Definition 2.1.5. A TGD σ is *frontier-guarded*, if it is of the form $\sigma : \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\ \Psi(\mathbf{Y}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms that contains an atom $r(\mathbf{Y})$ with $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subseteq \mathcal{V}$, $\mathbf{X} \cap \mathbf{Y} = \emptyset$, and $\Psi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms. The atom $r(\mathbf{Y})$ is the frontier-guard of σ.

Example 2.1.6. The TGDs σ_1, σ_2, and σ_3 from Examples 2.1.3 and 2.1.4 are all frontier-guarded, with $r(A, B, C)$ the frontier guard of σ_1 and σ_2, and $p(X, X)$ the frontier guard of σ_3. Note that σ_2 is not guarded, but it is frontier-guarded. The TGD σ_4 from Example 2.1.5 is not frontier-guarded.

Weakly Frontier-Guarded TGDs. A set of TGDs is *weakly frontier-guarded* (WYG) [5] if, for each TGD, there is an atom a in its body that contains all affected variables from the frontier of the TGD. Note that $\mathsf{YG} \subset \mathsf{WYG}$ and $\mathsf{WG} \subset \mathsf{WYG}$.

Definition 2.1.6. A set Σ of TGDs is *weakly-frontier-guarded*, if for each TGD $\sigma \in \Sigma$, it holds that all the variables of the frontier of σ that appear only at affected positions in the body of σ appear together in a body atom. This atom is called the weak-frontier-guard.

Example 2.1.7. The set of TGDs $\Sigma = \{s(X, Y) \wedge p(Z, Y) \rightarrow \exists W p(Y, W) \wedge s(W, X)\}$ is weakly frontier-guarded, with $s(X, Y)$ being the weak frontier-guard. The affected positions are $s[1]$ and $p[2]$ and the frontier variables are X and Y. The set of TGDs $\Sigma = \{s(X, Y) \wedge p(Y, Z) \rightarrow \exists W p(Z, W) \wedge s(W, X)\}$ is not weak frontier-guarded, since the affected positions are $s[1]$ and $p[2]$ and the frontier variables are X and Z.

Other guarded classes. Disconnected TGDs (denoted D) [6] have an empty frontier , but the body and the head may share constants. Note that $\mathsf{D} \subset \mathsf{YG}$.

Joint frontier guarded rules (JYG) [101] TGDs are a more general class of weakly-guarded and weakly-frontier-guarded, with WYG $\subset$ JYG. A TGD is *frontier-one* (denoted Y1) [7], if the size of the frontier is one. Glut variables may represent an overabundance of values, as opposed to the remaining, non-glut variables that can only represent finitely many values. Two classes aroused from this notion, namely *glut-guardeness* [101, 102] (denoted gG) and *glut-frontier-guarded* [101, 102] (denoted gYG)

2.1.4.2 Classes of TGDs based on Acyclicity

Each of the classes from below base their acyclicity on the construction of a graph. For example, weakly acyclic TGDs use a so-called position-dependency graph.

Acyclic TGDs.

Definition 2.1.7. A *position dependency graph* of a set of TGDs Σ has the nodes representing positions in predicates, i.e., the node (p, i) represents a position i in predicate p. For each TGD σ of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{Y}, \mathbf{Z})$ and each variable B in its body occurring in position (p, i), edges with origin (p, i) are built as follows: if $Y \in \mathbf{Y}$, there is an edge from (p, i) to each position of Y in head of σ; furthermore, for each $Z \in \mathbf{Z}$ occurring in position (q, j), there is a special edge from (p, i) to (q, j).

Definition 2.1.8. A set Σ of TGDs is *acyclic* if its position dependency graph does not contain a cycle. The corresponding class of TGDs is denoted A.

Example 2.1.8. Let $\sigma_1 : \mathit{proc}(V, W, X, Y) \rightarrow \exists\, Z\ s(W, Z) \wedge p(W, Y)$, $\sigma_2 : p(W, X) \wedge s(W, Y) \rightarrow t(Y, X)$, and $\sigma_3 : t(X, Y) \rightarrow p(X, Y)$ be TGDs. The set of TGDs $\Sigma = \{\sigma_1\}$ is acyclic, since we have no cycles, while $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ is not acyclic, since we have an cycle with the nodes $(t, 2)$ to $(p, 2)$.

Weakly acyclic TGDs. The next class deals with weakly acyclic TGDs, introduced in the context of data exchange [61].

Definition 2.1.9. A set Σ of TGDs is *weakly-acyclic*, and the associated class is denoted WA, if its position dependency graph does not contain a cycle through a special edge.

Note that weakly acyclic TGDs can be safely combined with EGDs, that they have a finite universal model, and that $\mathsf{A} \subset \mathsf{WA}$. Intuitively, infinity may result from the introduction of an existential variable in a given position which may lead to create another existential variable in the same position, hence an infinite number of existential variables. The position dependency graph helps us to avoid this from happening.

Example 2.1.9. Let $\sigma_1 : \mathit{proc}(V, W, X, Y) \rightarrow \exists\ Z\ s(W, Z) \wedge p(W, Y)$, $\sigma_2 : p(W, X) \wedge s(W, Y) \rightarrow t(Y, X)$, $\sigma_3 : t(X, Y) \rightarrow p(X, Y)$, and $\sigma_4 : \mathit{proc}(V, W, X, Y) \rightarrow \mathit{proc}(V, Y, X, W)$ be TGDs. The set of TGDs $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ is weakly acyclic, since there is no recursion with existential rule (even though we have recursion with Datalog rule, i.e., we have cycle from the nodes $(t, 2)$ to $(p, 2)$), while $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ is not weakly acyclic, since we have an existential rules cycle, i.e., the cycle with the nodes $(r, 2)$, $(r, 4)$, and $(s, 2)$.

Other acyclic classes. *Joint-acyclicity* [101] (denoted JA) is obtained by simply shifting the focus from positions to existential variables using the *existential dependency graph*, where the nodes are the existential variables occurring in rules. Three acyclicity notions that generalise JA are $\mathsf{MFA} \supset \mathsf{JA}$,

Another type of graph used is the *acyclic graph rule dependencies* [8] (denoted AGRD). Intuitively, AGRD introduces a rule dependency relation $\succ$ (or a graph of rules), for which $r_1 \succ r_2$ means that an application of rule r_1 on an instance can subsequently trigger an application of rule r_2. If the relation $\succ$ is acyclic, then no rule can trigger itself so the skolem chase terminates on an arbitrary instance. When all strongly connected components of the graph of rules dependencies have the property of being *weakly-acyclic sets of rules* (denoted WAGRD), then the forward chaining is finite

For $\mathsf{Adn}-\mathsf{WA}$ [82] one first rewrites a set of TGDs Σ into another set of TGDs Σ' by adorning the positions in the predicates that can contain infinitely many terms during the chase; then, one checks whether Σ' is WA.

If the TGDs have a *hypergraph-acyclic body* one can have the following classes: body acyclic frontier guarded [7] (denoted $\mathsf{ba}-\mathsf{YG}$) and *body acyclic frontier one* [7] (denoted $\mathsf{ba}-\mathsf{Y1}$).

2.1.4.3 Classes of TGDs based on Stickiness

Stickiness is inherently different from guardedness, and its central property is as follows: variables that appear more than once in a body (i.e., join variables) are always propagated (or "stick") to the inferred atoms.

Sticky Sets of TGDs.

Definition 2.1.10. *SMarking* takes a set Σ of TGDs as input and returns a set of marked TGDs where for each marked TGD, every body variable is either marked, or not. *SMarking* is defined inductively as the least fixpoint of the following:

- **Initial Marking Step.** For each TGD $\sigma \in \Sigma$ and each variable $V \in body(\sigma)$, if there exists an atom a in $head(\sigma)$ that does not contain V, mark V in σ.
- **Propagation Step.** For each TGD $\sigma \in \Sigma$ and each variable $V \in body(\sigma)$, if there exists a $\sigma' \in \Sigma$ such that $head(\sigma)$ and $body(\sigma')$ both contain a relation r and the positions where V appears in r in $head(\sigma)$ coincide exclusively with positions of marked variables in $body(\sigma')$, then mark V in σ.

Definition 2.1.11. A set of TGDs Σ is *sticky*, and the corresponding class is denoted S, if there is no TGD $\sigma \in SMarking(\Sigma)$ such that a marked variable occurs in $body(\sigma)$ more than once.

Example 2.1.10. Let $\sigma_1 : p(X,Y) \wedge s(Y,Z) \rightarrow \exists\, W\ r(W,Z,Y)$, $\sigma_2 : r(X,Y,Z) \rightarrow \exists W\ t(Z,W)$, and $\sigma_3 : r(X,Y,Z) \rightarrow \exists W\ t(X,W)$ be TGDs. The set of TGDs $\Sigma =$

$\{\sigma_1, \sigma_2\}$ is sticky, since the joined variable Y in σ_1 is propagated, while $\Sigma = \{\sigma_1, \sigma_3\}$ is not sticky, since the joined variable Y of σ_1 is not propagated in σ_3.

Weakly-Sticky TGDs. Weak stickiness is a relaxation of stickiness where only "harmful" variables are taken into account. A set of TGDs that enjoys weak stickiness is *weakly sticky*, and the associated class is denoted WS (intuitively, if a marked variable occurs more than once in a TGD body, then at least one of these positions has to be safe, i.e., only a finite number of terms can appear in this position during the forward chaining). Observe that $\mathsf{S} \subset \mathsf{WS}$.

Definition 2.1.12. A set of TGDs Σ is called *weakly sticky* if there is no TGD $\sigma \in SMarking(\Sigma)$ such that a marked variable V occurs in $body(\sigma)$ more than once and a position where V occurs is reachable from a cycle through a special edge in the position dependency graph of Σ.

Example 2.1.11. Let $\sigma_1 : p(X, Y) \wedge p(Y, Z) \rightarrow p(X, Z)$, $\sigma_2 : p(X, Y) \rightarrow s(X, Y)$, $\sigma_3 : s(X, Y) \rightarrow \exists Z\ s(Y, Z)$, and $\sigma_4 : p(X, Y) \rightarrow \exists Z\ p(Y, Z)$, be TGDs. The set of TGDs $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ is weakly-sticky, since in the relation p only a finite number of symbols occur (since there is no value invention) and the part of the chase that does not have sticky property (i.e., that has the relation p in it) is finite. The set of TGDs $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ is not weakly sticky, since the part of the chase that does not have sticky property (i.e., that has the relation p in it) is infinite.

Other sticky classes. Sticky join TGDs, denoted SJ, generalise S to allow linear TGDs. Weakly-sticky-join TGDs, denoted WSJ, generalise both WS and SJ.

2.1.4.4 Full TGDs.

Another key fragment of TGDs are *full* TGDs, i.e., TGDs without existentially quantified variables, and the corresponding class is denoted F. If one further assumes that full TGDs enjoy linearity, guardedness, stickiness, or acyclicity, then one obtains the classes LF, GF, SF, and AF, respectively.

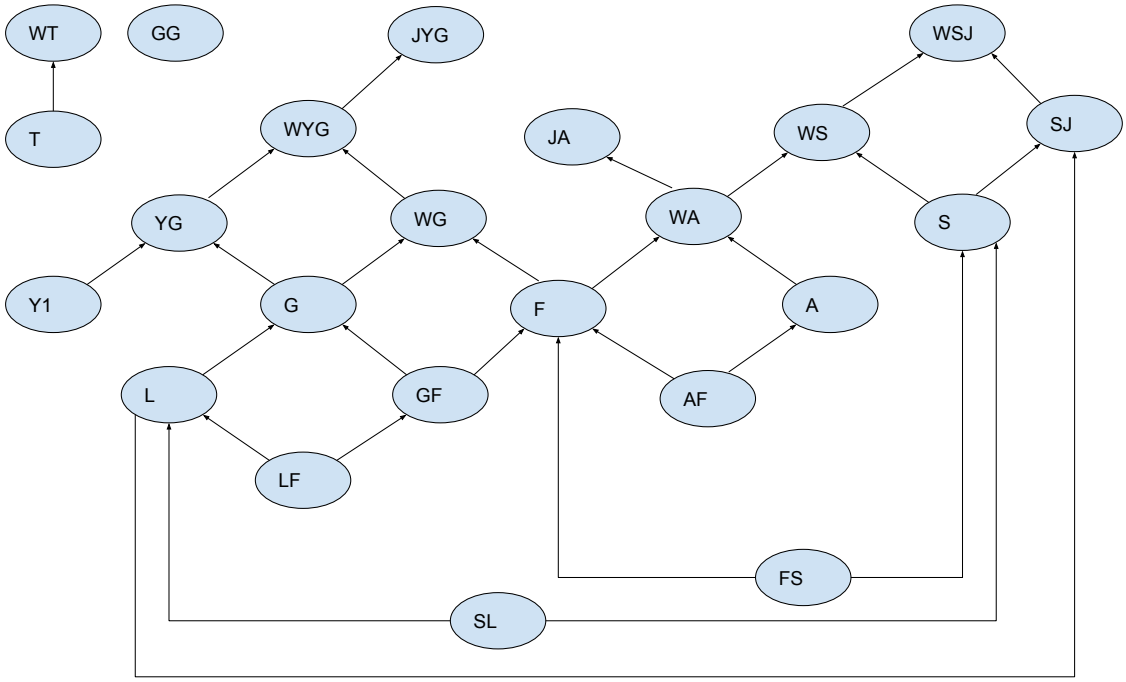


Figure 2.1: Classes hierarchy

2.1.4.5 Other classes

Tameness [76] (denoted T) merges guardedness and stickiness-join.

A set of TGDs is *shy* (denoted as H) [105] if during a Chase execution, nulls do not meet each other to join but only to propagate, with a null being propagated from a single atom only.

Figure 2.1 shows the relation between different classes, while Tables 2.2 to 2.4 presents the data, combined, *ba*-combined, and *fp*-combined complexity of CQ for *Datalog*$^{\exists}$ for the above introduced languages.

2.1.4.6 Relations between syntactic classes and semantic classes

Examples of the fes semantic classes are: F, Y1, Y1, WA. Examples of the bst semantic classes are: WYG and its subclasses, such as Y1 and G. Examples of fus classes are L and S.

Language	Data	Comb.	*ba*-comb.	*fp*-comb.
L	AC_0 [30]	PSPACE-c [32]	NP-c [88, 122]	NP-c [88, 122]
LF	AC_0 [30]	PSPACE-c [122]	NP-c [37, 122]	NP-c [37, 122]
AF	AC_0 [30]	PSPACE-c [32]	NP-c [37, 122]	NP-c [37, 122]
G	PTIME-c [30, 31]	2EXP-c [28, 29]	EXP-c [28, 29]	NP-c [28, 29]
WG	EXP-c [32]	2EXP-c [28, 29]	EXP-c [28, 29]	EXP-c [28, 29]
S	AC_0 [32, 35]	EXP-c [32, 35]	NP-c [122]	NP-c [122]
GF, SF	AC_0 [33]	EXP-c [122]	NP-c [37, 122]	NP-c [37, 122]
F	PTIME-c [48]	EXP-c [36]	NP-c [37, 122]	NP-c [37, 122]
A	AC_0 [30]	NEXP-c [122]	NEXP-c [122]	NP-c [122]
WA	PTIME-c [48, 61]	2EXP-c [33, 35]	2EXP-c [122]	NP-c [122]
WS, WSJ	PTIME-c [33]	2EXP-c [33]	2EXP-c [122]	NP-c [122]
T	PTIME-c [76]	2EXP-c [76]	EXP-c [76]	NP-c [76]

Table 2.2: Data, combined, *ba*-combined, and *fp*-combined complexity of CQ for $Datalog^{\exists}$ in different languages. On the complexity results columns, for a given complexity class $\mathcal{C}$

Language	Data	Comb.	*ba*-comb.
Y1,YG	PTIME-c [7]	2EXP-c [7, 101]	2EXP-c [7]
WYG	EXP-c [7]	2EXP-c [7]	2EXP-c [7]
gG, gYG	EXP-c [101]-hard	3EXP-c [101]	3EXP-c [101]
ba − Y1	PTIME-c [7]	EXP-c $[7]^H$	EXP-c [7]
ba − YG	PTIME-c [7]	2EXP-c [7]	EXP-c [7]

Table 2.3: Data, combined, and *ba*-combined complexity of CQ for $Datalog^{\exists}$ in different languages. On the complexity results columns, for a given complexity class $\mathcal{C}$

2.2 Preference Representation Languages

2.2.1 Qualitative Preferences

2.2.1.1 Properties of Binary Relations

A binary relation r on a set S is any subset of $\succeq \subseteq S \times S$. The relation r over the set S is an equivalence (EQ), a strict partial order (SPO), a weak order (WO), total order (TO), preorder (PR), or partial order (PO), as defined in Table 2.5. Preference relations ($\succ$ or $\succeq$) are binary relations that are either SPO, WO, TO, PR or PO. A relation $\succeq$ is cyclic if and only if its induced relation $\succ$ is cyclic, that is, there exists a chain of elements $o, \ldots, o'$ such that $o \succ \cdots \succ o' \succ o$.

Language	Data	Comb.
SJ	AC_0 [33]	EXP-C [33]
JA	PTIME-C [101]	2EXP-C [101]
JYG, JG	EXP-C [101]	2EXP-C [101]
H	PTIME-C [105]	EXP-C [105]
WAGRD	in PTIME-C [61]	in 2EXP-C [61]

Table 2.4: Data and combined complexity of CQ for $Datalog^{\exists}$ in different languages. On the complexity results columns, for a given complexity class $\mathcal{C}$

Table 2.5: Properties of a binary relation r on S

#	Property	Definition	EQ	SPO	WO	TO	PR	PO
1.	reflexive	$\forall x \in S: (x,x) \in r$	✓			✓	✓	✓
2.	irreflexive	$\forall x \in S: (x,x) \notin r$		✓	✓			
3.	symmetric	$\forall x,y \in S: (x,y) \in r \Rightarrow (y,x) \in r$	✓					
4.	asymmetric	$\forall x,y \in S: (x,y) \in r \Rightarrow (y,x) \notin r$		✓	✓			
5.	antisymm.	$\forall x,y \in S: (x,y) \in r \wedge (y,x) \in r \Rightarrow x = y$			✓	✓		
6.	transitive	$\forall x,y,z \in S: (x,y) \in r \wedge (y,x) \in r \Rightarrow (x,z) \in r$	✓	✓	✓	✓	✓	✓
7.	complete	$\forall x,y \in S: x \neq y \Rightarrow (x,y) \in r \vee (y,x) \in r$				✓		

2.2.1.2 Binary Relations and Preferences.

When I talk about preferences, I consider the relation to be expressed over a set of attributes $\mathcal{X}$. Each of the attribute X has a set of values, called its domain denoted $Dom(X)$ ($Dom(X) = \{x_1, \ldots, x_{|X|}\}$). For a subset U of $\mathcal{X}$, a value u of its domain, $Dom(U)$ is a function from U to $\bigcup_{X \in U} Dom(X)$ assigns to each variable $X \in U$ a value from $Dom(X)$. A partial assignment of U associates with each attribute a value from its domain, that is, any value $a \in Dom(U)$.

An *outcome* o (alternative or choice) associates with each attribute a value from its domain, that is, any value $o \in Dom(\mathcal{X})$. Note that the number of outcomes increases exponentially with the number of attributes. The outcome o is *dominated* by the outcome o' if $o' \succ_{\mathcal{X}} o$, and o is *directly dominated* by o', denoted $o \succ^d o'$, if $o \succ o'$, and there is no outcome o'' such that $o \succ o''$ and $o'' \succ o'$. If there is no

outcome o such that $o \succ o'$ than o' is *undominated*.

2.2.1.3 Semantics of Preferences

Let $\succ$ be a preference relation and lets assume that p is preferred to q (where p and q are partial assignments). There are multiple semantics for this statement:

- Strong semantics [168]: Any $(p \wedge \neg q)$ outcome is preferred w.r.t. $\succ$ to any $(q \wedge \neg p)$ outcome
- Ceteris paribus semantics [20]: Any $(p \wedge \neg q)$ outcome is preferred w.r.t. $\succ$ to an $(q \wedge \neg p)$ outcome, if the two outcomes have the same valuation over variables not appearing in $(p \wedge \neg q)$ and $(q \wedge \neg p)$
- Optimistic semantics [18]: At least one $(p \wedge \neg q)$ outcome is more preferred w.r.t. $\succ$ all $(q \wedge \neg p)$ outcomes
- Pessimistic semantics [12]: At least one $(q \wedge \neg p)$ outcome is less preferred w.r.t. $\succ$ to all $(p \wedge \neg q)$ outcomes
- Opportunistic semantics [161]: At least one $(p \wedge \neg q)$ outcome is preferred w.r.t. $\succ$ to a $(q \wedge \neg p)$ outcome

Example 2.2.1. Assume we have two attributes $\mathcal{X} = \{A, B\}$, $Dom(A) = \{a, a'\}$, and $Dom(B) = \{b, b'\}$. Therefore, the set of outcomes is $Dom(\mathcal{X}) = \{ab, a'b, ab', a'b'\}$. Suppose we say "I prefer a to a'" and we have the following relation $\{r(ab, a'b'), r(a'b', a'b), r(ab, a'b), r(a'b, ab'), r(ab', a'b)\}$. For the strong semantics or ceteris paribus semantics, the relation r does not satisfy "I prefer a to a'" since, we do not have $r(ab', a'b')$. For the optimistic semantics, the relation holds, since we have $r(ab, a'b')$ and $r(ab, a'b)$, the same for pessimistic one, since $r(ab, a'b)$ and $r(ab', a'b)$ and opportunistic semantics. ∎

For more information regarding semantics, I refer the reader to [94]. For the rest of this thesis, I focus on the ceteris paribus semantic and the optimistic semantics, because they both induce a unique partial order over the set of outcomes.

2.2.2 CP-Theories

CP-theories are a natural, concise, and flexible representation of qualitative preferences – along with their subclasses (such as CP-nets), they are widely used to represent and reason with qualitative preferences. The formalism is based on the 'ceteris paribus" semantics (see above). I now recall CP-theories from [168].

Syntax. A *conditional preference* φ on $\mathcal{X}$ has the form

$$\varphi = u\colon x \succ x'[W], \tag{2.2}$$

where $U \subseteq \mathcal{X}$, $u \in Dom(U)$, $X \in \mathcal{X} \setminus U$, $x, x' \in Dom(X)$, and $W \subseteq \mathcal{X} \setminus (U \cup \{X\})$. For $\varphi = u\colon x \succ x'[W]$, I write u_φ, U_φ, x_φ, x'_φ, W_φ, and T_φ to denote u, U, x, x', W, and $\mathcal{X} - (U \cup \{X\} \cup W)$, respectively.

Intuitively, such φ means that, given u and $t \in Dom(T)$, one prefers x to x', irrespective of the value of W where $T = \mathcal{X} \setminus (U \cup \{X\} \cup W)$.

A *conditional preference theory* Γ (or *CP-theory*) on $\mathcal{X}$ is a finite set of conditional preferences on $\mathcal{X}$.

Semantics. Interpretations $\mathcal{I}$ of CP-theories are strict total orders on $Dom(\mathcal{X})$. For conditional preferences $\varphi = u\colon x \succ x'[W]$, let $\varphi^\star = \{(tuxw, tux'w') \mid t \in Dom(T_\varphi), w, w' \in Dom(W)\}$. The outcome $tux'w'$ is defined as a *worsening flip* from $tuxw$ (with respect to φ), while φ is a compact representation of φ^*.

An interpretation $\mathcal{I}$ *satisfies* (or *is a model of*) φ, denoted $\mathcal{I} \models \varphi$, if $\mathcal{I} \supseteq \varphi^\star$. $\mathcal{I}$ *satisfies* (or *is a model of*) a CP-theory Γ if it satisfies all $\varphi \in \Gamma$.

A CP-theory Γ is *consistent* if it has a model. Let $\Gamma^\star$ denote the transitive closure of $\bigcup_{\varphi\in\Gamma} \varphi^\star$. Then, for all outcomes o and o', it holds that $o \succ o'$ iff $(o, o') \in \Gamma^\star$.

Pairs $(\alpha, \beta) \in \varphi^*$ represent a preference for α over β, and φ is a compact representation of the preference information φ^*. Let us assume that preferences are transitive, so the order $\succ_\Gamma$ induced on $Dom(\mathcal{X})$ by Γ is the transitive closure of Γ^*.

CP-nets [20] (resp., TCP-nets [25]) can be represented via conditional preferences of the form $u : x \succ x'[W]$ with $W = \emptyset$ (resp., $|W| \leq 1$).

Example 2.2.2. Alice would like to rent a property and she prefers a house (h) over an apartment (a) (statement (1) below), even though she can live also in the latter. Since she owns a car, she would always prefer a property with garage over one with pool irrespective of the property type or location (2). Alice knows that location l_1 has better gyms than l_2, and l_1 is further from her work than l_2. Therefore, if a property has a garage (g), then she would prefer the location l_1 over l_2, since she can commute by car (3). Still, if there is an apartment that has swimming pool (p), she would prefer location l_2 over l_1 (4). If Alice finds a house, she would prefer to be in location l_1 over l_2, since she is thinking to build her gym for training (5).

Based on the above description of Alice's preferences, let us introduce three variables T (type: house or apartment), L (location: l_1 or l_2), and F (feature: garage or pool) with the domains $Dom(T) = \{h, a\}$, $Dom(L) = \{l_1, l_2\}$, and $Dom(F) = \{g, p\}$, respectively, and one can model Alice's preferences by the CP-theory

$$\begin{aligned}\Gamma \;=\; &\{(1)\,\top\colon h \succ a[\emptyset],\ (2)\,\top\colon g \succ p[\{T, L\}],\\ &(3)\,g\colon l_1 \succ l_2[\emptyset],\ (4)\,ap\colon l_2 \succ l_1[\emptyset],\ (5)\,h\colon l_1 \succ l_2[\emptyset]\},\end{aligned}$$

which has the following two models:

$$ghl_1 \succ gal_1 \succ ghl_2 \succ gal_2 \succ phl_1 \succ phl_2 \succ pal_2 \succ pal_1,$$
$$ghl_1 \succ ghl_2 \succ gal_1 \succ gal_2 \succ phl_1 \succ phl_2 \succ pal_2 \succ pal_1.$$

For example, the outcome $o = ghl_1$ represents a house with garage situated in location l_1. ∎

2.2.2.1 Complexity Results of Dominance testing

Deciding dominance between two outcomes in a standard CP-theory is PSPACE-complete [72].

Given this negative result, research focused on different ways to overcome it. The first way is to identify fragments of CP-theories for which dominance testing is tractable. For example, for standard polytree CP-nets [20] (of bounded node in-degree), dominance between two outcomes can be decided in polynomial time [20], while for acyclic polynomially connected CP-nets it is NP-hard [20]. When the number of rules of generalised CP-nets [20] is parametrized, dominance testing is tractable [100]. For an unconditional fragment of TCP-nets [25], the complexity of dominance testing is polynomial [146]. Another tractable subclass of CP-nets, which is characterised by what the authors call monotonic variables is presented in [174].

The second way is to find an approximation of CP-theories/nets [54, 93, 108, 169, 171] for which preference inference can be done in polynomial time.

The third way is to detect languages similar to CP-theories that have better complexity results. For example conditional importance (CI) nets' [22] complexity of dominance falls down to P for the case of precondition-free singletons. Another language is hierarchical CP-networks [129] with a lighter ceteris-paribus principle, where dominance testing can be solved in polynomial time in the size of the hierarchical CP-networks. There are preference formalisms with polynomial dominance testing, such as the ones in [40, 171].

2.2.3 PrefDatalog$^{\pm}$

Let us recall the PrefDatalog$^{\pm}$ language introduced in [115], which is a generalisation of Datalog+/– with preferences.

In the following, I denote by Δ_{Ont}, $\mathcal{V}_{Ont}$, and $\mathcal{R}_{Ont}$ the infinite set of constants, the infinite set of variables, and the finite set of predicates, respectively, of standard Datalog+/– ontologies, as described in the previous section. These sets give rise to the corresponding Herbrand base $\mathcal{H}_{Ont}$.

A *preference relation* in this case is any binary relation $\succ \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$ that is a strict partial order, which are irreflexive and transitive relations.

Consider a set of *preference formulas* (here, one can consider a slight generalisation of the proposal by [39]); that are first-order expressions of the form pf: $C(a, b)$ defining a preference relation $\succ_{pf}$ on $\mathcal{H}_{Ont}$ as follows: $a \succ_{pf} b$ if $C(a, b)$. We call $C(a, b)$ the *condition* of pf, denoted $cond(pf)$. The preference relation defined via a set of preference formulas P is denoted $\succ_P$.

Example 2.2.3. Consider again the ontology $O = (D, \Sigma)$ from Example 2.1.1. The preferences statements of (1) and (2) of Example 2.2.2 can be expressed as follows:

$$C_1 : \mathit{property}(X, T_1, L, C) \succ \mathit{property}(Y, T_2, L, C) \text{ if } T_1 = \mathit{house} \wedge$$
$$T_2 = \mathit{apt} \wedge X \neq Y \wedge \mathit{propertyFeature}(Y, F) \wedge \mathit{propertyFeature}(X, F)$$
$$C_2 : \mathit{property}(X, T_1, L_1, C) \succ \mathit{property}(Y, T_2, L_2, C) \text{ if } \mathit{propertyFeature}(X, F_1)$$
$$\wedge \mathit{propertyFeature}(Y, F_2) \wedge \mathit{feature}(F_2, g) \wedge \mathit{feature}(F_1, p) \wedge X \neq Y$$

■

A preference-based Datalog$^{\pm}$ ontology (PrefDatalog$^{\pm}$ ontology, or knowledge base) is of the form $KB = (O, P)$, with O a Datalog$^{\pm}$ ontology, and P a set of preferences formulas over atoms in O. Intuitively, for the semantics, the consequences of a knowledge base $KB = (O, P)$ are computed in terms of the chase for the classical Datalog$^{\pm}$ ontology O, and the set of preference formulas P describes the preference relation over pairs of the ground atoms in O.

Example 2.2.4. Let $KB = (O, U_1)$ be a PrefDatalog$^{\pm}$ ontology, with $O = (D, \Sigma)$ from Example 2.1.1 and U_1 from Example 2.2.3. We can see that $KB \not\models \mathit{property}(t_2) \succ \mathit{property}(t_4)$ (does not satisfy C_2) and $KB \models \mathit{property}(t_1) \succ \mathit{property}(t_2)$. ■

In the case of guarded PrefDatalog$^{\pm}$ (i.e, the underlying ontology is guarded) answering top most preferred answers (skyline answers) for DAQs can be computed in polynomial time in the data complexity [115].

For CQs Q, and when the relational schema of O and Σ are fixed, deciding if the set of skyline answers answer to Q are non-empty is Σ_2^p-complete [115]. In case

the ontology is FO-rewritable (a class C of TGDs is first-order(FO) rewritable if for every set of TGDs Σ in C, and for every BCQ Q, there exists a first-order query Q_Σ such that for every database instance D, it holds $D \cup \Sigma \models Q$ iff $D \models Q_\Sigma$), one can materialise the necessary part of the ontology into the query, and calculate the top answers just over the database.

Part II

Qualitative Preferences for the Semantic Web

Chapter 3

Single User Qualitative Preferences

Users find it easier to express their preferences using qualitative relations than scores [53].

In this chapter, I introduce PP_Datalog$^{\pm}$ [116] ontologies, which integrate Datalog$^{\pm}$ ontologies with both qualitative preferences over the consequences of the ontologies and uncertainty (probabilities over them). The work in this area has been on defining new preference combination operators, which produce a preference relation, given a general SPO that models preferences, and a score-based SPO that models uncertainty. I analyse in more detail PP_Datalog$^{\pm}$ by evaluating the two existing operators, and proposing two new specific operators. For example, one operator allows the user to choose how much influence the probabilistic model has on the output. Another example is to use as base the user's preferences, and to use the probabilistic model as a secondary source of "advice". Note that all previous work on extending Datalog$^{\pm}$ with uncertainty [74] does not deal with preferences.

The rest of this chapter is organised as follows. Section 3.1 introduces the general preference and probabilistic models that are used in this work. Section 3.1.1 presents the syntax and semantics of PP_Datalog$^{\pm}$, and different combination operators

in Section 3.1.2. Section 3.2 introduces the semantic properties of the proposed operators. Section 3.3 contains algorithms for the implementation of four preference combination operators, presents skyline and k-rank queries, a basic algorithm to answer them, and proves its correctness and running time, showing that under certain conditions, k-rank queries can be answered in polynomial time in the data complexity. In Section 3.4, I provide a real-world application, and report on an implementation along with extensive experimental results, respectively. Section 3.5 concludes the chapter.

3.1 Formalisms

The PP_Datalog$^{\pm}$ [116] framework combines preferences and probabilistic uncertainty in Datalog$^{\pm}$ ontologies for a single user, where the preferences of every user are expressed as strict partial orders (SPO: irreflexive, antisymmetric and, transitive binary relations; see Table 2.5).

Assuming that more probable answers are in general more preferable, one asks how to rank answers to a user's queries, since the preference model may be in conflict with the preferences induced by the probabilistic model—the need thus arises for preference combination operators.

3.1.1 PP_Datalog$^{\pm}$: Syntax and Semantics

In this section, I introduce the PP_Datalog$^{\pm}$ language, an extension of Datalog$^{\pm}$ with both a preference model and a probabilistic model; this formalism is based on the one first presented in [115], which extends Datalog$^{\pm}$ with preferences (but not probabilities). To this end, I assume the following sets giving rise to the logical languages for ontologies, preferences, and probability models: Δ_{Ont}, Δ_{Pref}, and Δ_M are finite sets of constants, $\mathcal{R}_{Ont}$, $\mathcal{R}_{Pref}$, and $\mathcal{R}_M$ are finite sets of predicate names such that $\mathcal{R}_M \cap \mathcal{R}_{Ont} = \emptyset$, and $\mathcal{V}_{Ont}$, $\mathcal{V}_{Pref}$, and $\mathcal{V}_M$ are infinite sets of variables. In the following, I assume w.l.o.g. that $\mathcal{R}_{Pref} \subseteq \mathcal{R}_{Ont}$, $\Delta_{Pref} \subseteq \Delta_{Ont}$, and $\mathcal{V}_{Pref} \subseteq \mathcal{V}_{Ont}$. These sets give rise to corresponding *Herbrand bases* consisting of all possible ground

atoms that can be formed, denoted with $\mathcal{H}_{Ont}$, $\mathcal{H}_{Pref}$, and $\mathcal{H}_{M}$, respectively. Clearly, I have $\mathcal{H}_{Pref} \subseteq \mathcal{H}_{Ont}$, meaning that preference relations are defined over a subset of the possible ground atoms.

Preference Models. A *preference relation* is any binary relation $\succ \subseteq \mathcal{H}_{Pref} \times \mathcal{H}_{Pref}$. In this chapter, I am interested in strict partial orders (SPOs), which are irreflexive and transitive relations—I consider these to be the minimal requirements for a preference relation to be useful in the applications that I envision. One way of specifying such a relation that is especially compatible with my approach is the preference formula framework of [39]. In this work, I assume the existence of a general *preference model* P specifying a preference relation over a subset of $\mathcal{H}_{Ont}$, denoted $\succ_P$; in general, one can treat $\succ_P$ as a set of ordered pairs. When a preference relation $\succ$ is induced by an assignment of a numeric score to each element in such a way that $a_1 \succ a_2$ if $score(a_1) > score(a_2)$, it is said to be $\succ$ *score-based*.

Example 3.1.1. Following the topic of Example 2.1.1, a preference relation might be specified by Bob over the *place* atoms -places he would like to rent; an example of such a relation is shown in Figure 3.1, where I assume the transitive closure of the graph depicted. For instance, this preference relation can be obtained from a user with the following preferences:

- Prefers luxurious place over economy one; $place(p_2)$ and $place(p_4)$ are incomparable with each other.
- For economy class, prefers location l_3 over l_1 and l_2; we thus have that $place(p_6)$ is preferred over $place(p_1)$, $place(p_3)$, and $place(p_5)$.
- As a particular preference, the user prefers $place(p_1)$ to $place(p_5)$. ∎

Probabilistic Models. For modelling uncertainty, let us assume the existence of a probabilistic model M that represents a probability distribution $\Pr_M$ over some set $X = \{X_1, \ldots, X_n\}$ of Boolean variables such that there is a 1-to-1 mapping

from X to the set of all ground atoms over $\mathcal{R}_M$ and Δ_M. Examples of the type of probabilistic models that I assume in this work are Markov logic and Bayesian networks. The probabilistic extension of Datalog$^{\pm}$ adopted here was first introduced in [114]; probabilistic query answering (without preferences) in a version of this model using Markov logic was also studied in [74].

Substitutions. A substitution is a mapping from variables to variables or constants. Two sets S and T *unify* via a substitution θ if $\theta S = \theta T$, where θA denotes the application of θ to all variables in all elements of A (here, θ is a unifier). A *most general unifier* (mgu) is a unifier θ such that for all other unifiers ω, there is a substitution σ such that $\omega = \sigma \circ \theta$.

Definition 3.1.1. Let M be a probabilistic model. Then, a *(probabilistic) annotation* λ relative to M is a (finite) set of expressions of the form $\langle A_i = x_i \rangle$, where: (i) A_i is an atom over $\mathcal{R}_M$, $\mathcal{V}_M$, and Δ_M; and (ii) $x_i \in \{0, 1\}$. A probabilistic annotation is *valid* if for any two different expressions $\langle A = x \rangle, \langle B = y \rangle \in \lambda$, there does not exist a substitution that unifies A and B.

Intuitively, a probabilistic annotation is used to describe the class of events, in which the random variables in a probabilistic model M are compatible with the settings of the random variables described by λ, i.e., each X_i has the value x_i. A *probabilistic scenario* is a valid probabilistic annotation λ, for which $|\lambda| = |X|$ and all $\langle A = x_i \rangle \in \lambda$ are such that A is ground. Let us use $scn(M)$ to denote the set of scenarios in M.

Example 3.1.2. Continuing with the running example, suppose an online real estates rating system is used to derive a probabilistic model that assigns probabilities specifying how likely it is that the user will like each movie in the knowledge base. The system in question could be aggregating information from user feedback, their friends' feedback, and any other information available to it. Thus, the system informs the user of the probability associated with each *place* atom; this could be done by

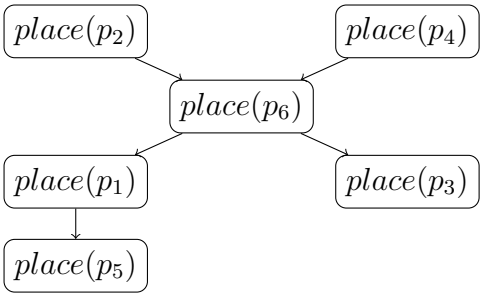


Figure 3.1: Preference relation $\succ_P$.

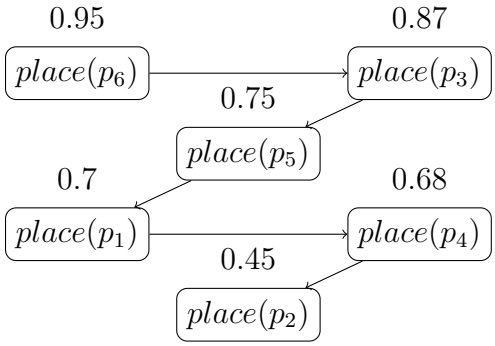


Figure 3.2: Probabilistic model $\succ_M$.

extending the ontology from Example 2.1.1 by replacing such atoms in the database with formulas of the form: $\mathit{place}(P) : \{\mathit{enjoy}(P) = 1\}$, where $\mathit{enjoy}(P)$ denotes the probabilistic event that place P will be enjoyed by the user — I am assuming here that the model is designed for a single user. Figure 3.2 gives an example of such a probability assignment (derived as explained in the semantics section below), along with the preference relation in graph form that is induced by these values assuming that higher probabilities are more preferable. ∎

Preference Combination Operators. As seen in Figures 3.1 and 3.2, the particular challenge encountered in PP_Datalog$^{\pm}$ ontologies is that the preference model yields a certain precedence relation that might be in disagreement with the one induced by the probabilistic model. To address this situation, I make use of *preference combination operators*, which take two preference relations, and produce a third one satisfying two basic properties.

Definition 3.1.2. Let $\succ_P$ be an SPO and $\succ_M$ be a score-based preference relation. A *preference combination operator* $\otimes(\succ_M, \succ_P)$ yields a relation $\succ^{\otimes}$ such that: (i) $\succ^{\otimes}$ is an SPO, and (ii) if $a_1 \succ_P a_2$ and $a_1 \succ_M a_2$, then $a_1 \succ^{\otimes} a_2$.

The two properties required by Definition 3.1.2 are the minimal required to produce a "reasonable" combination of the two relations; as it is shown below, particular implementations may satisfy further desirable properties.

We are now ready to define PP_Datalog$^{\pm}$ ontologies.

Definition 3.1.3. A *PP_Datalog*$^{\pm}$ *ontology* (or *PP-KB*) is of the form $KB = (O, P, M, \otimes)$, where O is a Datalog+/– ontology with probabilistic annotations, P is a preference model, M is a probabilistic model with Herbrand bases $\mathcal{H}_{Ont}$, $\mathcal{H}_{Pref}$, and $\mathcal{H}_M$, respectively, such that $\mathcal{H}_{Pref} \subseteq \mathcal{H}_{Ont}$, and $\otimes$ is a preference combination operator. If O is a guarded Datalog$^{\pm}$ ontology, then KB is *guarded*.

In the following, I discuss the semantics of PP-Datalog+/– ontologies, and formally define the kinds of queries that I address in this chapter.

Semantics. Let $KB = (O, P, M, \otimes)$ be PP_Datalog$^{\pm}$ ontology; the semantics of PP_Datalog$^{\pm}$ arises as a direct combination of the semantics of Datalog$^{\pm}$ and that of the preference and probabilistic models. Relative to the probabilistic model, $\Pr_{KB}(a) = p$ if

$$p = \sum_{\lambda \in scn(M), O_\lambda \models a} \Pr_M(\lambda).$$

Let us refer to the score-based preference relation induced by $\Pr_M$ as the *probabilistic preference relation* associated with KB, and denote it with $\succ_M$. Let $\succ^{\otimes} = \otimes(\succ_M, \succ_P)$; $KB \models a_1 \succ^{\otimes} a_2$ if:

1. $O \models a_1$ and $O \models a_2$; and
2. $a_1 \succ^{\otimes} a_2$.

Intuitively, the consequences of a knowledge base $KB = (O, P, M, \otimes)$ are computed in terms of the classical consequences of the Datalog$^{\pm}$ ontology O, and the preference combination operator yields a preference relation over pairs of atoms in $\mathcal{H}_{Ont}$.

Skyline and k-Rank Queries. In this chapter, I am interested in computing answers for skyline queries [16]. It is a well-known class of queries that can be issued over preference-based formalisms, and the iterated computation of skyline answers that allows us to assign a *rank* to every atom. In the following, I focus on a specific kind of classical queries, called disjunctive atomic queries (DAQs), which

are disjunctions of atoms. I do not focus on conjunctive queries because of the intractability of the preference language is based on (see Section 2.2.3).

Definition 3.1.4. Let $KB = (O, P, M, \otimes)$ be a PP_Datalog$^{\pm}$ ontology, $\succ^{\otimes} = \otimes(\succ_M, \succ_P)$, and $Q(\mathbf{X}) = q_1(\mathbf{X}_1) \vee \cdots \vee q_n(\mathbf{X}_n)$ be a DAQ. Then, a *skyline answer* to Q relative to $\succ^*(= (\succ^{\otimes})^{\star})$ is any θq_i entailed by O such that no θ' exists with $O \models \theta' q_j$ and $\theta' q_j \succ^{\otimes} \theta q_i$, where θ and θ' are ground substitutions for the variables in $Q(\mathbf{X})$.

To obtain an ordered list of answers to queries under transitive relations, I adopt the *iterated skyline* approach proposed in [39], which simply assigns a *rank* to each answer by iterating the following procedure: (1) assign the rank k to the skyline answers; and (2) remove from consideration all answers of rank k, increment k by one, and go back to (1). As a generalisation of classical top-k queries, I adopt here *k-rank queries*; answers to such queries are simply k-tuples of answers sorted by rank. In the following, I use $rank(a, \succ_M)$ to denote the rank assigned to atom a by relation $\succ_M$.

Intuitively, for DAQs, both kinds of answers can be seen as atomic consequences of O that satisfy the query: the skyline answers can be seen as sets of atoms that are not dominated by any other such atom, while k-rank answers are k-tuples sorted according to the preference relation. I refer to these as answers in *atom form*.

In the next section, I will present algorithms for implementing two particular preference combination operators, which (as it is shown through a series of properties) produce a new preference relation $\succ^{\otimes}$ that is useful in answering k-rank queries that adequately reflect both the initial preferences expressed by the user as well as the fact that higher probability answers are more desirable.

3.1.2 Preference Combination Operators

In this section, I study two families of preference combination operators and the semantic properties of several particular instantiations. Section 3.3.2 provides an algorithm that uses these operators for answering k-rank queries to PP_Datalog$^{\pm}$

ontologies in polynomial time in the data complexity (modulo the cost of computing probabilities with respect to the probabilistic model M).

3.1.2.1 An Egalitarian Class of Combination Operators

The first class of operators is called *egalitarian*, in principle allows the combination of its input SPOs to closely resemble either one.

A General Preference Combination Operator. This operators was originally presented in [116]. I analyse its properties more in detail.

Algorithm 1 An algorithm for combining an arbitrary SPO with a weak order in the form of a score-based preference relation.

Input: (i) SPO $\succ_P$,
(ii) score-based preference relation $\succ_M$ over $\mathcal{H}_{Ont}$, and
(iii) $t \in [0, 1]$.
Output: Preference relation $\succ_t^{\otimes} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.
Called by: Algorithm 6.

```
1: function ComPrefsGen(≻_M, ≻_P, t)
2:     Initialise ≻_t^⊗ as a copy of ≻_P;
3:     for all pairs (a_i, a_j) ∈≻_P do
4:         if score(a_j) − score(a_i) > t then // scores of M
5:             change (a_i, a_j) to (a_j, a_i) in ≻_t^⊗;
6:             if associated graph of ≻_t^⊗ is cyclic then
7:                 change (a_j, a_i) to (a_i, a_j) in ≻_t^⊗;
8:     return transitiveClosure(≻_t^⊗).
```

Algorithm ComPrefsGen in Algorithm 1 implements a preference combination operator using a value $t \in [0, 1]$ that allows the user to choose how much influence the probabilistic model has on the output preference relation; I use $\succ_t^{\otimes}$ to denote the output relation. The algorithm iterates through all pairs of elements in $\succ_P$. If (i) $\succ_M$ disagrees with $\succ_P$, (ii) the difference in score is greater than t, and (iii) inserting the pair according to $\succ_M$ does not produce a cycle, then the pair is inserted in reverse order into the output; otherwise, the output contains the same pair as $\succ_P$. Finally, the algorithm outputs the transitive closure of this relation. Note that if $t = 0$, then the probability-induced relation has precedence; at the other end of the spectrum, if $t = 1$, then $\mathsf{ComPrefsGen}(\succ_M, \succ_P, t) = \succ_P$ (these properties are presented formally

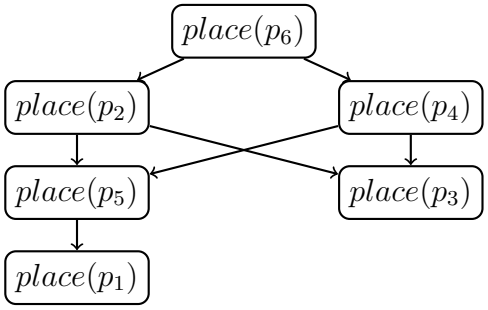


Figure 3.3: ComPrefsGen($\succ_M, \succ_P, 0$).

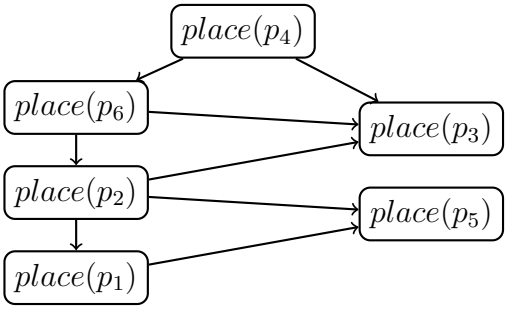


Figure 3.4: ComPrefsGen($\succ_M, \succ_P, 0.3$)

Rank	$\succ_P$	$\succ_M$	$\succ_0^{\otimes}$= ComPrefsGen($\succ_M, \succ_P, 0$)	$\succ_{0.3}^{\otimes}$
1	$[X/p_2]$, $[X/p_4]$	$[X/p_6]$	$[X/p_6]$	$[X/p_4]$
2	$[X/p_6]$	$[X/p_3]$	$[X/p_2]$, $[X/p_4]$	$[X/p_6]$
3	$[X/p_1]$, $[X/p_3]$	$[X/p_5]$	$[X/p_5]$, $[X/p_5]$	$[X/p_2]$
4	$[X/p_5]$	$[X/p_1]$	$[X/p_1]$	$[X/p_1]$, $[X/p_3]$
5	–	$[X/p_4]$	–	$[X/p_5]$
6	–	$[X/p_2]$	–	–

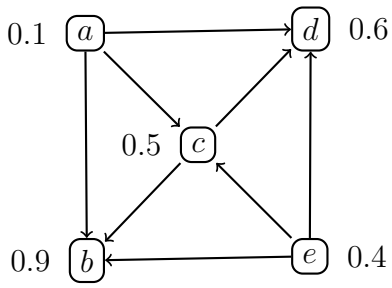
Table 3.1: Answers to query *place*(X) according to their rank relative to the original preference relation (Figure 3.1), probabilistic model (Figure 3.2), and the combination of the two relations using ComPrefsGen algorithm with $t = 0$ and $t = 0.3$.

in Theorem 3.2.3). The following is an example of how the algorithm works.

Example 3.1.3. Consider again the running example. Figures 3.3 and 3.4 show the result of running ComPrefsGen over the two preference relations with $t=0$ and $t=0.3$ — note that, for clarity, the transitive closures are not shown in these figures. Table 3.1 shows the result of computing the rank via the iterative computation of the skyline answers for the three preference relations. As we can see, the user's original preferences ($\succ_P$) are preserved to a greater extent in $\succ_{0.3}^{\otimes}$ than in $\succ_0^{\otimes}$: the answers in the former have a difference in rank of at most 1, whereas those in the latter have a difference of 2 (out of a total of 4) in multiple cases. This reflects the greater influence that $\succ_P$ has in the result for $t=0.3$ as compared to $t=0$. ∎

An Egalitarian Operator Based on Rank. In this section, I study a different kind of combination operator, which is based on a trade-off between generality and the properties that can be proved about the result, originally introduced in [116].

The following example shows a shortcoming of the ComPrefsGen operator in that



Order	Edge	$t = 0.1$	$t = 0.05$
1	(c, d)	No (t)	Yes
2	(e, d)	No (cycle)	Yes
3	(e, c)	No (t)	Yes
4	(a, c)	Yes	No (cycle)
5	(a, d)	Yes	Yes
6	(a, b)	Yes	No (cycle)
7	(c, b)	Yes	No (cycle)
8	(e, b)	Yes	No (cycle)

Figure 3.5: User preferences for Example 3.1.4 (left), and the results of applying Algorithm ComPrefsGen for two different values of t (right). In the table, "yes" and "no" means whether or not the edge is reversed in the result—in the negative cases, the reason is given in parenthesis.

it fails to exhibit a kind of monotonicity that might be expected when the two input relations disagree on a given pair (a_1, a_2). In particular, if the operator chooses the order imposed by $\succ_M$, then it may not continue to do so for smaller values of t. On the other hand, if the operator chooses the order imposed by $\succ_P$, then it may also fail to do so for greater values of t.

Example 3.1.4. Consider the preference relation in Figure 3.5 (top), where the directed edges between nodes describe a relation $\succ_P$, and the numbers beside each node define score-based relation $\succ_M$. The table at the bottom specifies what happens when the two are input to the ComPrefsGen algorithm, and the edges are inspected in the specified order, for two values of t. Note row 4, where edge (a, c) becomes (c, a) under $t = 0.1$, but stays unaltered under $t = 0.05$, and row 6, where edge (a, b) stays unaltered under $t = 0.05$, but becomes (b, a) under $t = 0.1$. ∎

As shown in Example 3.1.4, the cause of this kind of unpredictability is the potential presence of cycles when merging preference relations. Algorithm ComPrefsRank(Algorithm 2) avoids this issue by combining the two relations into a score-based one based on the rank of each atom relative to each input relation. The algorithm takes as input an integer binary function that combines the two ranks, and assigns it to the atom in the output relation — this function can be as simple as

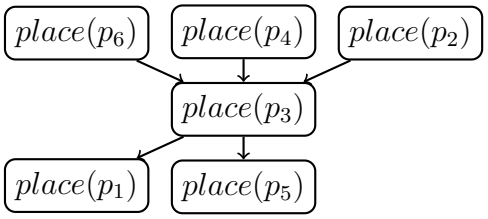


Figure 3.6: $\succ_{min}^{rank}$.

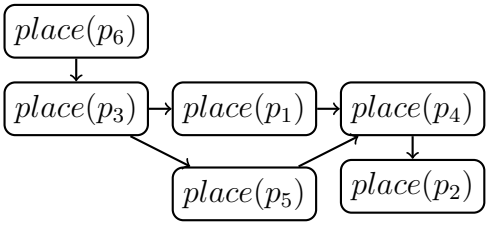


Figure 3.7: $\succ_{max}^{rank}$

`min`, `max`, or a more complex function that takes into account how much the user would like to base the new rank on one relation or the other. A detailed treatment of such functions is outside the scope of this chapter.

The rank for a score based preference relation $\succ_M$ is computed as follows. The highest score has the rank 1, the second highest score rank 2 and so on. If two atoms have the same score, than they have the same rank.

The rank of SPO $\succ_P$ is computed as follows. The undominated atoms in the relation $\succ_P$ have the rank 1. Than one removes the atoms of rank from $\succ_P$, obtaining $\succ'_P$. The undominated atoms in the relation $\succ'_P$ have the rank 2 and so on.

Algorithm 2 Algorithm for combining an SPO with a score-based preference relation, based on rank.

Input: (i) SPO $\succ_P$,
(ii) score-based preference relation $\succ_M$ over $\mathcal{H}_{Ont}$, and
(iii) integer binary function f such that $f(x, y) \in \{x, y\}$.
Output: Preference relation $\succ_f^{rank} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.
Called by: Algorithm 6.

1: **function** ComPrefsRank($\succ_M, \succ_P, f$)
2: Initialise $\succ_f^{rank}$ as an empty preference relation over $\subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$;
3: Initialise r_M and r_P as an empty mapping from $\mathcal{H}_{Ont}$ to integers;
4: **for all** atoms $a \in \mathcal{H}_{Pref}$ **do**
5: set $r_M(a)$ to be the rank of a in $\succ_M$; //rank while sorting by score
6: set $r_P(a)$ to be the rank of a in $\succ_P$; //rank of topological order
7: add a to $\succ_f^{rank}$ with score $f(r_M(a), r_P(a))$;
8: **return** transitiveClosure($\succ_f^{rank}$).

The following example shows the behaviour of Algorithm ComPrefsRank over the running example.

Example 3.1.5. Consider once again the running example. Figures 3.6 and 3.7

Rank	Relation $\succ_P$	Relation $\succ_M$	Operator $\succ_{min}^{rank}$	Operator $\succ_{max}^{rank}$
1	$[X/p_2], [X/p_4]$	$[X/p_6]$	$[X/p_2]$, $[X/p_4]$,$[X/p_6]$	$[X/p_6]$
2	$[X/p_6]$	$[X/p_3]$	$[X/p_3]$	$[X/p_3]$
3	$[X/p_1]$, $[X/p_3]$	$[X/p_5]$	$[X/p_5]$, $[X/p_1]$	$[X/p_1]$, $[X/p_5]$
4	$[X/p_5]$	$[X/p_1]$	–	$[X/p_4]$
5	–	$[X/p_4]$	–	$[X/p_2]$
6	–	$[X/p_2]$	–	–

Table 3.2: Answers to query $place(X)$ according to their rank relative to the original preference relation (Figure 3.1), probabilistic model (Figure 3.2), and the combination of the two relations using ComPrefsRank algorithm with min and max functions.

Atom	r_P = rank in $\succ_P$	r_M= rank in $\succ_M$	$min(r_P, r_M)$	$max(r_P, r_M)$
$place(p_2)$	1	6	1	6
$place(p_4)$	1	5	1	5
$place(p_6)$	2	1	1	2
$place(p_3)$	3	2	2	3
$place(p_1)$	3	4	3	4
$place(p_5)$	4	3	3	4

Table 3.3: Score-based SPOs induced. Min and Max are calculated over Rank ($\succ_P$) and Rank ($\succ_M$).

show the result of running ComPrefsRank over the two different input functions (*min* and *max*) — note that, for clarity, the transitive closures are not shown in these figures. Table 3.2 shows the results of computing the rank via the iterative computation of the skyline answers for the three preference relations while Table 3.3 describes the process carried out.

As we can see, the user's original preferences ($\succ_P$) are preserved to a greater extent in $\succ_{min}^{rank}$ than in $\succ_{max}^{rank}$, with $\succ_{min}^{rank}$= ComPrefsRank($\succ_M, \succ_P, min$), and $\succ_{max}^{rank}$= ComPrefsRank($\succ_M$,$\succ_P, max$). ∎

3.1.2.2 A User-Biased Class of Combination Operators

The second class of operators I study is called *user-biased*; the main characteristic of this kind of operators is that they base the result SPO on the user's preferences, and use the probabilistic model as a secondary source of "advice".

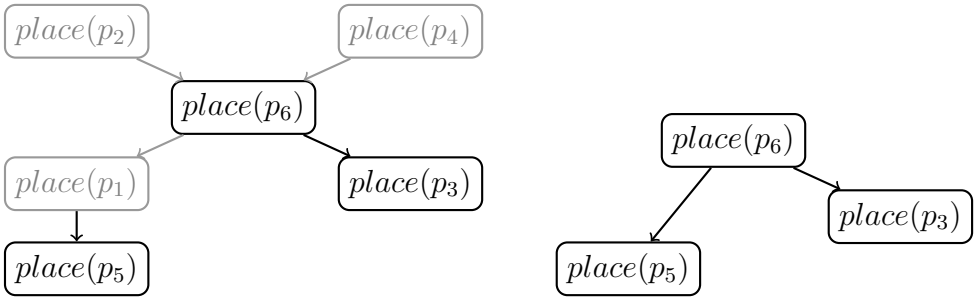


Figure 3.8: The original SPO $\succ_P$ (left), where atoms whose probabilities are less than 0.75 according to $\succ_M$ have been shaded, and the result of applying Algorithm ComPrefsPT (right).

An Operator Based on Probability Thresholds. One way of obtaining an SPO that is biased by the user's input is to remove from consideration any elements that have probability below a given threshold — all previously existing relationships between atoms remain the same. The following is an example of how the ComPrefsPT operator works.

Algorithm 3 Algorithm for performing a user-biased combination of an SPO with a score-based preference relation, based on a given probability threshold.

Input: (i) SPO $\succ_P$,
(ii) score-based preference relation $\succ_M$ over $\mathcal{H}_{Ont}$, and
(iii) probability threshold $p \in [0, 1]$.
Output: Preference relation $\succ_p^{pt} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.
Called by: Algorithm 6.

```
1: function ComPrefsPT(≻_M, ≻_P, p)
2:     Initialise ≻_p^pt as a copy of ≻_P;
3:     for all atoms a ∈ H_Pref do
4:         if score(a) < p then //score of M
5:             remove element a from ≻_p^pt;
6:     return transitiveClosure(≻_p^pt).
```

Example 3.1.6. Returning to the running example, Figure 3.8 shows the original SPO $\succ_P$, and the result of applying the preference combination operator implemented by Algorithm ComPrefsPT with threshold 0.75, called $\succ_{0.75}^{pt}$. Note that relations between remaining atoms in the original SPO still hold; for instance, $place(p_6) \succ_{0.7}^{pt} place(p_5)$, which was true relative to $\succ_P$ due to transitivity. ∎

Algorithm 4 Algorithm for performing a user-biased combination of an SPO with a score-based preference relation, based on sorting skylines.

Input: (i) SPO $\succ_P$,
(ii) score-based preference relation $\succ_M$ over $\mathcal{H}_{Ont}$.
Output: Preference relation $\succ_p^{pt} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.
Called by: Algorithm 6.

```
1:  function ComPrefsSort(≻_M, ≻_P)
2:      Initialise ≻^sort as a copy of ≻_P;
3:      k ← 1;
4:      repeat
5:          currentRank ← all elements of rank k relative to ≻^sort;
6:          k ← k + 1;
7:          Sort the elements of currentRank in dec. order of scores of M;
8:          for all i ∈ {1, 2, ..., length(currentRank) − 1} do
9:              add preference (currentRank[i], currentRank[i + 1]) to ≻^sort;
10:     until curr-rank is empty
11:     return transitiveClosure(≻^sort).
```

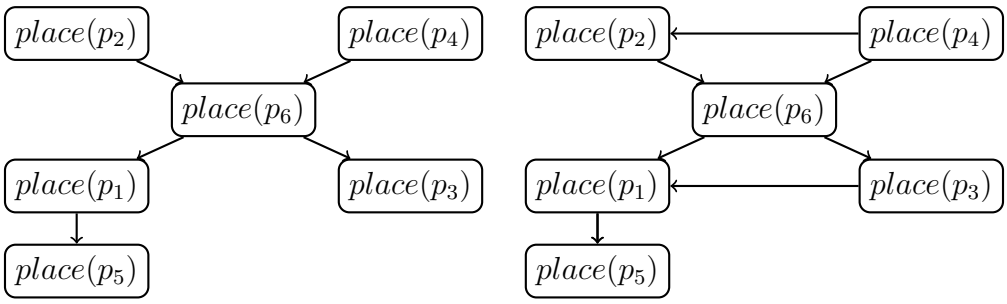


Figure 3.9: The original SPO $\succ_P$ (left), and the result of applying Algorithm ComPrefsSort (right).

A Skyline Sorting Operator. Another possibility for a user-biased operator is to keep the same base structure given by the user, and make use of the probability-based preference relation to remove to the furthest extent possible the indifference present among elements of the same rank, i.e., in each iterated skyline. Algorithm ComPrefsSort (see Algorithm 4) implements such an operator. The following is an example of how the algorithm works.

Example 3.1.7. Consider again the running example. Figure 3.9 shows $\succ_P$ (left), and the SPO resulting from Algorithm ComPrefsSort. Note that the two relations are very similar, except that the indifference between $place(p_2)$ and $place(p_4)$, and the

one between $place(p_1)$ and $place(p_3)$ have been replaced by the ordering suggested by $\succ_M$. ∎

3.2 Semantic Properties

This section proves that each of the proposed combination operators return an SPO and if there is no disagreement between the probabilistic model and the preference model, then the combination operator should keep the ordering. These are two reasonable properties that one would expect. Other examples of proved properties related with extreme cases and the presence of cycles.

3.2.1 An Egalitarian Class of Combination Operators

3.2.1.1 A General Preference Combinator Operator

The following result shows that ComPrefsGen encodes indeed a preference combination operator (cf. Definition 3.1.2).

Theorem 3.2.1. Let $\succ_P$ be an SPO, $\succ_M$ be a score-based preference relation, $t \in [0, 1]$, and $\succ_t^{\otimes} = \mathsf{ComPrefsGen}(\succ_M, \succ_P, t)$. Then, for any value of t,

(i) $\succ_t^{\otimes}$ is an SPO, and

(ii) if $a_1 \succ_P a_2$ and $a_1 \succ_M a_2$, then $a_1 \succ_t^{\otimes} a_2$.

Proof. (i) One must show that $\succ_t^{\otimes}$ is antisymmetric, irreflexive, and transitive. Let $\succ'$ be the intermediate result of the ComPrefsGen operator just before computing the transitive closure. By hypothesis, $\succ_P$ is irreflexive, and (as $\succ'$ cannot contain any new self-edges) this property is preserved in $\succ'$. Similarly, as $\succ_P$ is antisymmetric, and ComPrefsGen checks for cycles before changing edges, there will never be two edges (a, b) and (b, a)—antisymmetry therefore vacuously holds. Finally, by construction, $\succ_t^{\otimes}$ is the transitive closure of $\succ'$; since this operation cannot add cycles, we can conclude that $\succ_t^{\otimes}$ is an SPO.

(ii) A necessary condition for ComPrefsGen to change the order of a pair in $\succ_P$ is that the pair in $\succ_M$ be reversed. As by hypothesis this is not the case, the statement follows. □

Let us now study the properties enjoyed by the output of the ComPrefsGen algorithm. The following theorem states that the output for $t=0$ is invariant to changes in the assigned scores, as long as the order is the same. This is an important property, since it implies that approximation algorithms can be used to compute the probability values that induce the $\succ_M$ relation — as long as the relative order of the atoms is guaranteed to be correct, there is no need to compute the exact values when $t = 0$.

Theorem 3.2.2. Let $\succ_P$ be an SPO, and $\succ_M$ and $\succ_{M'}$ be score-based preference relations. If $\succ_M = \succ_{M'}$, then:

$$\mathsf{ComPrefsGen}(\succ_M, \succ_P, 0) = \mathsf{ComPrefsGen}(\succ_{M'}, \succ_P, 0).$$

Proof. For $t=0$, the algorithm checks (in line 3) if there exist pairs (a_i, a_j) in $\succ_P$ such that $\Pr(a_j) > \Pr(a_i)$. Since $\succ_M = \succ_{M'}$, by hypothesis, $\Pr(a_j) > \Pr(a_i)$ relative to $\succ_M$ iff this is the case relative to $\succ_{M'}$, and thus the outputs in both cases must be identical. □

The next theorem shows the behaviour of the operator for the extreme values of parameter t.

Theorem 3.2.3. Let $\succ_P$ be an SPO, and $\succ_M$ be a score-based preference relation. Then,

(i) $\mathit{rank}(a, \mathsf{ComPrefsGen}(\succ_M, \succ_P, 0)) = \mathit{rank}(a, \succ_M)$, for every atom a; and

(ii) $\mathsf{ComPrefsGen}(\succ_M, \succ_P, 1) = \succ_P$.

Proof. (i) For $t = 0$, the output relation is the result of comparing all pairs in $\succ_P$ and replacing them with the order imposed on their elements by $\succ_M$. Thus, given that $\succ_P$ is transitive, the sequence of atoms giving rise to the rank of each atom is reordered in the resulting relation according to the rank of each element in $\succ_M$, and the statement follows.

(ii) Direct consequence of the condition in line 3 of Algorithm 1: since the difference in probabilities can never be greater than 1, the output relation is a copy of $\succ_P$. □

Finally, the following theorem studies a property that is analogous to the "sensitivity postulates" presented in [179], [177, pp. 12–14,63–64]. Note, however, that these postulates were designed to combine two score-based preference relations, whereas my setting is more general — the "faithfulness" postulate in [177] is already guaranteed by the definition of preference combination operator.

Theorem 3.2.4. Let $KB = (O, P, M, \mathsf{ComPrefsGen})$ be a PP_Datalog$^{\pm}$ ontology, Q be a DAQ, $k \geq 0$, and $\succ_t^{\otimes} = \mathsf{Com\ PrefsGen}(\succ_M, \succ_P, t)$, with $t \in [0, 1]$. For every atom a that does not belong to any k-rank answer to Q over KB relative to $\succ_t^{\otimes}$, there exists a probabilistic model M' and $t' \in [0, 1]$ such that a belongs to some k-rank answer to Q over $KB' = (O, P, M', \mathsf{ComPrefsGen})$ relative to $\succ'_{t'} = \mathsf{ComPrefsGen}(\succ_{M'}, \succ_P, t')$.

Proof. By hypothesis, there exist atoms $a_1, \ldots, a_n$ such that $a_i \succ_t^{\otimes} a$. Now, let $t' = 0$, and set the probabilities in M' such that $\Pr_{M'}(a) > \Pr_{M'}(a_i)$, for $1 \leq i \leq n$. By construction, $a \succ'_{t'} a_i$, and the statement follows. □

3.2.1.2 An Egalitarian Operator Based on Rank

The following theorem states that ComPrefsRank satisfies the conditions for being a preference combination operator for certain functions.

Theorem 3.2.5. Let $\succ_P$ be an SPO, $\succ_M$ be a score-based preference relation, f be an integer binary function such that $f(x, y) \in [x, y]$, and $\succ_f^{rank}$= ComPrefsRank($\succ_M$, $\succ_P, f$). Then:

(i) $\succ_f^{rank}$ is an SPO; and

(ii) If $a_1 \succ_P a_2$, $a_1 \succ_M a_2$, and $f \in \{min, max, avg\}$, then $a_1 \succ_f^{rank} a_2$.

Proof. (i) Direct consequence, since $\succ_f^{rank}$ is a score-based preference relation. (ii) Let $r_1^P = rank(a_1, \succ_P)$, $r_2^P = rank(a_1, \succ_P)$, $r_1^M = rank(a_1, \succ_M)$, and $r_2^M = rank(a_1, \succ_M)$. The hypotheses imply that $r_1^P < r_2^P$ and $r_1^M < r_2^M$. Clearly, $f(r_1^P, r_1^M) < f(r_2^P, r_2^M)$ for $f \in \{min, max, avg\}$, and thus $a_1 \succ_f^{rank} a_2$. □

Analysing the proof of Theorem 3.2.5, we can see that the result also holds for variations of the functions considered — as long as the condition $f(r_1^P, r_1^M) < f(r_2^P, r_2^M)$ holds.

Another property of ComPrefsRank that can be shown is the analogous of Theorem 3.2.2; since the hypothesis implies that ranks relative to $\succ_M$ and $\succ_{M'}$ are equal, the result clearly holds. Finally, the following theorem discusses properties related to the postulates from [177] for this algorithm:

Theorem 3.2.6. Let $KB = (O, P, M,$ ComPrefsRank$)$ be a PP_Datalog$^{\pm}$ ontology, Q be a DAQ, $k \geq 0$, and $\succ_f^{\otimes}$ = Com- PrefsRank($\succ_M, \succ_P, f$). Then:

(i) Let $f \in \{min, max, avg\}$, M' be a probabilistic model such that for some ground atom a, we have $\Pr_{KB'}(a) \leq \Pr_{KB}(a)$ and $\Pr_{KB'}(a') = \Pr_{KB}(a')$ for every ground atom $a' \neq a$; let $KB' = (O, P, M',$ ComPrefsRank$)$. If a does not belong to any k-rank answer to Q over KB relative to $\succ_f^{\otimes}$, then a does not belong to any k-rank answer to Q over KB' relative to $\succ'_f$= ComPrefsRank($\succ_{M'}, \succ_P, f$).

(ii) Given the setup in part (i), if a belongs to some k-rank answer to Q over KB relative to $\succ_f^{\otimes}$ and $\Pr_{KB'}(a) \geq \Pr_{KB}(a)$, then a also belongs to some k-rank answer to Q over KB' relative to $\succ'_f$= ComPrefsRank($\succ_{M'}, \succ_P, f$).

(iii) For every atom a that does not belong to any k-rank answer to Q over KB relative to $\succ_f^{\otimes}$, there exists a probabilistic model M' and f' such that, a belongs to some k-rank answer to Q over $KB' = (O, P, M', \mathsf{ComPrefsRank})$ relative to $\succ'_{f'} = \mathsf{ComPrefsRank}(\succ_{M'}, \succ_P, f')$.

Proof. (i) $\Pr_{KB'}(a) \leq \Pr_{KB}(a)$ implies that we have $\mathit{rank}(a, \succ_{M'}) \leq \mathit{rank}(a, \succ_M)$, and the result from Theorem 3.2.5 can be used to show the result.

(ii) Analogous to (i): $\Pr_{KB'}(a) \geq \Pr_{KB}(a)$ implies that $\mathit{rank}(a, \succ_{M'}) \geq \mathit{rank}(a, \succ_M)$.

(iii) Using $f = \mathit{min}$, the rest of the proof is analogous to that of Theorem 3.2.4. □

3.2.2 A User-Biased Class of Combination Operators

3.2.2.1 An Operator Based on Probability Thresholds

The following theorem shows that ComPrefsPT satisfies the conditions stated in Definition 3.1.2:

Theorem 3.2.7. Let $\succ_P$ be an SPO, $\succ_M$ be a score-based preference relation, $p \in [0, 1]$, and $\succ_p^{pt} = \mathsf{ComPrefsPT}(\succ_M, \succ_P, p)$. Then,

(i) $\succ_p^{pt}$ is an SPO; and

(ii) If $a_1 \succ_P a_2$, $a_1 \succ_M a_2$, $\mathit{score}(a_1, \succ_M) \geq p$, and
$\mathit{score}(a_2, \succ_M) \geq p$, then $a_1 \succ_p^{pt} a_2$.

Proof. (i) Direct consequence, since $\succ_p^{pt} \subseteq \succ_P$.

(ii) The two conditions $\mathit{score}(a_1, \succ_M) \geq p$ and $\mathit{score}(a_2, \succ_M) \geq p$ imply that a_1 and a_2 appear in $\succ_p^{pt}$; therefore, since $a_1 \succ_P a_2$ and $\succ_p^{pt} \subseteq \succ_P$, the statement follows. □

Let us now see if the properties described in Theorem 3.2.6 hold for ComPrefsPT.

Theorem 3.2.8. Let $KB = (O, P, M, \mathsf{ComPrefsPT})$ be a PP_Datalog$^{\pm}$ ontology, Q be a DAQ, $p \in [0, 1]$, and $\succ_p^{\otimes} = \mathsf{ComPrefsPT}(\succ_M, \succ_P, p)$. Then:

(i) Let M' be a probabilistic model such that for some ground atom a, we have $\Pr_{KB'}(a) \leq \Pr_{KB}(a)$ and $\Pr_{KB'}(a') = \Pr_{KB}(a')$ for every ground atom $a' \neq a$; let $KB' = (O, P, M', \mathsf{ComPrefsPT})$. If a does not belong to any k-rank answer to Q over KB relative to $\succ_p^{\otimes}$, then a does not belong to any k-rank answer to Q over KB' relative to relation $\succ'_p = \mathsf{ComPrefsPT}(\succ_{M'}, \succ_P, p)$.

(ii) Given the setup in part (i), if a belongs to some k-rank answer to Q over KB relative to $\succ_p^{\otimes}$ and $\Pr_{KB'}(a) \geq \Pr_{KB}(a)$, then a also belongs to some k-rank answer to Q over KB' relative to $\succ'_p = \mathsf{ComPrefsPT}(\succ_{M'}, \succ_P, p)$.

(iii) For every atom a that does not belong to any k-rank answer to Q over KB relative to $\succ_p^{\otimes}$, there exists a probabilistic model M' and $p' \in [0, 1]$ such that a belongs to some k-rank answer to Q over $KB' = (O, P, M', \mathsf{ComPrefsPT})$ relative to $\succ'_{p'} = \mathsf{ComPrefsPT}(\succ_{M'}, \succ_P, p')$.

Proof. (i) Clearly, if $\Pr_{KB'}(a) \leq p$ then the result holds. Otherwise, since a does not belong to any k-rank with respect to $\succ_p^{\otimes}$, no other atom's status relative to the threshold changes, and ComPrefsPT does not use the probability values to produce its output, the statement follows.

(ii) The fact that a belongs to some k-rank answer implies that $\Pr_{KB}(a) \geq p$ and thus $\Pr_{KB'}(a) \geq p$. The rest of the proof is analogous to the second part of (i).

(iii) Using $p = \Pr_{KB}(a)$, and changing the probabilities of all atoms b such that $b \succ_P a$ to $\Pr_{KB}(a) - \epsilon$, for some $\epsilon \in [0, 1]$, it holds $rank(a, \succ'_{p'}) = 1$, and the statement follows. □

3.2.2.2 A Skyline Sorting Operator

The following theorem shows that ComPrefsSort implements indeed a preference combination operator.

Theorem 3.2.9. Let $\succ_P$ be an SPO, $\succ_M$ be a score-based preference relation, and $\succ^{sort} = \mathsf{ComPrefsPT}(\succ_M, \succ_P)$. Then:

(i) $\succ^{sort}$ is an SPO; and

(ii) If $a_1 \succ_P a_2$ and $a_1 \succ_M a_2$, then $a_1 \succ^{sort} a_2$.

Proof. (i) Since $\succ^{sort}$ is a copy of $\succ_P$ with the addition of preference pairs between elements that were previously unrelated, and there is no possibility of introducing cycles, the statement follows.

(ii) By construction, if $a_1 \succ_P a_2$ then also $a_1 \succ^{sort} a_2$. □

The following theorem states some properties of ComPrefsSort. Note that the property from Theorem 3.2.4 (also property (iii) in Theorems 3.2.6 and 3.2.8) does not hold for ComPrefsSort; this is because the operator does not allow the user's preference relation to change so drastically depending on the probabilistic model.

Theorem 3.2.10. Let $KB = (O, P, M, \mathsf{ComPrefsSort})$ be a PP_Datalog$^{\pm}$ ontology, Q be a DAQ, and $\succ_t^{\otimes} = \mathsf{ComPrefs\ Sort}(\succ_M, \succ_P)$. Then:

(i) Let M' be a probabilistic model such that for some ground atom a we have $\Pr_{KB'}(a) \leq \Pr_{KB}(a)$ and $\Pr_{KB'}(a') = \Pr_{KB}(a')$ for every ground atom $a' \neq a$; let $KB' = (O, P,\ M', \mathsf{ComPrefsSort})$. If a does not belong to any k-rank answer to Q over KB relative to $\succ_t^{\otimes}$, then a does not belong to any k-rank answer to Q over KB' relative to relation $\succ' = \mathsf{ComPrefsSort}(\succ_{M'}, \succ_P)$.

(ii) Given the setup in part (i), if a belongs to some k-rank answer to Q over KB relative to $\succ_t^{\otimes}$ and $\Pr_{KB'}(a) \geq \Pr_{KB}(a)$, then a also belongs to some k-rank answer to Q over KB' relative to $\succ'$= $\mathsf{ComPrefsSort}(\succ_{M'}, \succ_P)$.

Proof. (i) Since the result of ComPrefsSort only differs from $\succ_P$ in that elements of the same rank relative to $\succ_P$ are ordered with respect to $\succ_M$, all elements dominating a in $\succ_t^{\otimes}$ still do so in $\succ'$, and the statement follows.

(ii) Analogous to (i). □

3.3 Algorithms and Complexity

This section studies a basic algorithm for answering k-rank disjunctions of atomic queries, and showed that under certain conditions, k-rank queries can be answered in polynomial time in the data complexity, which is the same complexity as answering traditional preference-based queries in relational databases. Moreover, I have provided upper bounds of the complexity results for each of the operators.

3.3.1 Complexity of Combination Algorithms

I analyse the complexity of all combination algorithms introduced above. I begin by identifying and discussing the main tasks that the algorithms perform. In the rest of this section, let us assume that SPOs are represented as directed graphs; the size of such a graph G is determined by the number n of nodes in the graph (i.e., the number of atoms in the SPO), and the number e of edges in the graph (i.e., the number of pairs in the preference relation).

Transitive closure. All four algorithms compute as the last step the transitive closure of the preference relation that results from combination; this is required to return an SPO. The best known algorithms for computing the transitive closure are the ones from Warshall [167] and Warren [166], which have $O(n^3)$ as worst case running time. Clearly, in all algorithms, the dominant term is $O(n^3)$, contributed by the transitive closure. However, as shown in Section 7.4, if this last step is not required, then the running time of the operators in the worst case are quite different.

Cycle detection. The combination algorithm ComPrefsGen performs a check for potential cycles every time an edge from the original SPO needs to be inverted. The procedure to find a cycle in a directed graph can be done with a depth-first search of the graph, and checking for a back edge (edges that point from a node to one of its ancestors). The actual running time of this procedure depends on the representation of the graph, e.g., with adjacency lists, the worst case running time is $O(n + e)$.

Computing probabilities. The cost of computing probabilities relative to a probabilistic model M can range from polynomial time (such as in approximation algorithms, or tractable models like polytrees [96] or tractable Markov logic [52]) to #P-complete (such as in general Markov logic [144] or Bayesian networks). Furthermore, in some cases, it may be assumed that the probability of all ground atoms can be done offline, and stored in a look-up table, which can be consulted in constant time. This is the case for the experimental evaluation, as scores from IMDB yield probabilities that can be easily computed and added to the database.

3.3.1.1 Egalitarian Combination Algorithms

This section gives an analysis of the running time of the egalitarian combination algorithms.

Theorem 3.3.1. Let $\succ_P$ be an SPO, $\succ_M$ be a score-based preference relation, and S be the time required to compute the score of any atom relative to $\succ_M$. Then, ComPrefsGen can be done in time $O(n \cdot S + n^3 + e^2)$, and ComPrefsRank in time $O(n \cdot S + n^3)$, where n (resp., e) is the number of nodes (resp., edges) of $\succ_P$ represented as a directed graph.

Proof. Observe first that computing the probability of each node in M is possible in $O(n \cdot S)$.

Then, as for ComPrefsGen, this operator inspects every edge in the input SPO, computing the values of the probabilities of the endpoints, and checking for potential cycles. This part of the algorithm takes time $O(e) \cdot O(n + e)$. The last step computes the transitive closure of the updated graph that represents the new combined preference relation, which can be done in $O(n^3)$. Overall, ComPrefsGen can be done in time $O(n \cdot S + n^3 + e^2)$.

As for ComPrefsRank, the ranks of each atom relative to both preference relations are computed. For the SPO, this requires time $O(n + e)$; for the preference relation based on M, this can be done by sorting the nodes by probability value, and then

traversing that sorted list, this requires time $O(n \cdot \log(n))$. Then, the algorithm computes for each atom a score based on a function that combines the previously computed ranks of that atom; I assume that this function can be computed in constant time — therefore, this adds an $O(n)$ term. Finally, the transitive closure of the resulting relation is computed, adding a cost of $O(n^3)$. Overall, ComPrefsRank can be done in time $O(n \cdot S + n^3)$. □

As $\succ_P$ describes a preference relation over the consequences of an ontology, the following corollary states the data tractability of the above algorithms.

Corollary 3.3.2. Let $KB = (O, P, M, \otimes)$ be a guarded PP_Datalog$^{\pm}$ ontology, $\mathcal{H}_{Ont}$ the Herbrand base for O, $\succ_P$ an SPO, $\succ_M$ a score-based preference relation over $\mathcal{H}_{Ont}$, and Q a disjunction of atomic query. Then, if $\Pr_M(a)$ can be computed or approximated in polynomial time in the data complexity for any a with $KB \models a$, then ComPrefsGen and ComPrefsRank run in polynomial time in the data complexity.

Proof. For guarded Datalog$^{\pm}$ ontologies, the necessary finite part of the chase required to answer a query is of polynomial size in the data complexity [31], and it is precisely this structure that gives rise to $\succ_P$. By the hypothesis that probabilities are computed in polynomial time, and the result in Theorem 3.3.1, the statement follows. □

3.3.1.2 User-Biased Combination Algorithms

In this section, I provide an analysis of the running time of the user-biased combination algorithms.

Theorem 3.3.3. Let $\succ_P$ be an SPO, $\succ_M$ be a score-based preference relation, and S be the time required to compute the score of any atom relative to $\succ_M$. Then, ComPrefsPT and ComPrefsSort run in time $O(n \cdot S + n^3)$, where n is the number of nodes of $\succ_P$ represented as a directed graph.

Proof. Let $e = |\succ_P|$. Observe first that computing the probability of each node in M is possible in $O(n \cdot S)$.

Then, as for ComPrefsPT, this operator inspects every node, checking if its associated probability given by M is below the input threshold and, if so, it deletes the node and all of its incoming and outgoing edges. Since I assume that the cost of deleting edges is constant, this operation is possible in time $O(n + e)$. Finally, the transitive closure is computed, which is possible in $O(n^3)$. It thus follows that ComPrefsPTcan overall be done in $O(n \cdot S + n^3)$.

As for ComPrefsSort, this operator iteratively computes a skyline of the input SPO, and sorts the elements in the skyline to produce the new preference relation. There are two possible worst case scenarios for this algorithm: (i) the input relation contains $O(n + e)$ iterated skylines, or (ii) the input relation contains $r \in O(1)$ number of iterated skylines. For case (i), we have $O(n + e)$ iterated skylines, and there is a constant number of nodes that belong to each one; thus, we get a running time of $O(n + e)$. For the second case, we require $\sum_{i=1}^{r} O(n \cdot \log(n))$ operations, which yields a running time of $O(n \cdot \log(n))$. Finally, the transitive closure adds an $O(n^3)$ term, and thus ComPrefsSort can overall also be done in $O(n \cdot S + n^3)$. □

Analogous to the analysis for egalitarian operators, the following corollary states the data tractability.

Corollary 3.3.4. Let $KB = (O, P, M, \otimes)$ be a guarded PP_Datalog$^{\pm}$ ontology, $\mathcal{H}_{Ont}$ be the Herbrand base for O, $\succ_P$ be an SPO, $\succ_M$ be a score-based preference relation over $\mathcal{H}_{Ont}$, and Q be a CQ. Then, if $\Pr_M(a)$ can be computed or approximated in polynomial time in the data complexity for any a such that $KB \models a$, ComPrefsPT and ComPrefsSort run in polynomial time in the data complexity.

Proof. Analogous to the proof of Corollary 3.3.2. □

3.3.2 Answering k-Rank Queries

As discussed above, in this chapter, I am interested in computing the rank of the answers to queries by means of the iterated computation of its skyline answers [16].

I now present a general algorithm to do so, and then analyse its correctness as well as its running time when used in conjunction with either the ComPrefsGen or ComPrefsRank algorithm.

The Algorithm k-Rank (see Algorithm 6) begins by computing the combination of the two preference relations in the PP_Datalog$^{\pm}$ ontology and then calls the iteratedSkyline function introduced in Algorithm 5, which returns a k-answer.

The iteratedSkyline function first computes the necessary finite part of the chase relative to Q. The main `while` loop of Algorithm 5 iterates through the process of computing the skyline answers to Q relative to this new relation by using a computeSkyline subroutine (that can be implemented by means of a linear-time scan of C), updating the result by appending these answers in arbitrary order, and removing the atoms in the result from C. Once the loop is finished, the algorithm returns the first k results, as the last iteration might add superfluous elements.

Example 3.3.1. Consider again the running example and the query $Q = place(X)$, and $k = 4$. Using the results shown in the various examples above, one can obtain the following k-rank answers to Q (in atom form) depending on the preference combinator operators used:

- ComPrefsGen with $t = 0.3$: $\langle place(p_4),\ place(p_6),\ place(p_2),\ place(p_1)\rangle$;
- ComPrefsRank with $f = max$: $\langle place(p_6),\ place(p_3),\ place(p_1),\ place(p_5)\rangle$;
- ComPrefsPT with $p = 0.75$: $\langle place(p_6),\ place(p_3),\ place(p_5)\rangle$; and
- ComPrefsSort: $\langle place(p_4),\ place(p_2),\ place(p_6),\ place(p_3)\rangle$.

Analysing the differences among the different answers, one can see some characteristics of each operator. For instance, both ComPrefsGen with $t = 0.3$ and ComPrefsSort include one of the user's favourite places in the first position, though ComPrefsSort includes both of the top elements with respect to $\succ_P$ first and ComPrefsGen puts $place(p_6)$ before $place(p_2)$ given the latter's lower probability score.

Now consider the results yielded by ComPrefsRank with *max* and ComPrefsPT with $p = 0.75$. The former takes a pessimistic stance, and therefore the very low rank of the user's top choices pushes them outside the final rank. Similarly, the low scores of these elements make them unavailable to be chosen (no matter what value k has) by ComPrefsPT, since they do not surpass the threshold value. ∎

Algorithm 5 An algorithm for computing a k-rank answer to DAQ Q relative to $\succ$.

Input: (i) $KB = O(D, \Sigma)$,
(ii) DAQ $Q(\mathbf{X})$
(iii) SPO $\succ$, and
(iv) $k \geq 0$.
Output: k-rank answer $\langle a_1, \ldots, a_{k'} \rangle$ to Q, with $k' \leq k$.
Called by: Algorithms 6 and 8.

```
1: function iteratedSkyline(KB, Q, ≻, k)
2:     Initialise Res as an empty vector of ground atoms;
3:     C ← computeChase(O, Q); // Section 2.1.1.3
4:     i ← k;
5:     while i > 0 do
6:         S ← computeSkyline(C, Q, ≻);
7:         Append S to Res (in arbitrary order);
8:         Remove S from C;
9:         i ← i − |S|;
10:    return truncate(Res, k).
```

Algorithm 6 An algorithm for computing a k-rank answer to DAQ Q relative to the composition of $\succ_P$ and $\succ_M$.

Input: (i) $KB = (O, P, M, \otimes)$,
(ii) DAQ $Q(\mathbf{X})$ and
(iii) $k \geq 0$.
Output: k-rank answer $\langle a_1, \ldots, a_{k'} \rangle$ to Q, with $k' \leq k$.
Calls: One of the Algorithms 1 to 4 encoded in $\otimes$, and Algorithm 5.

```
1: function k-Rank(KB, Q, k)
2:     Initialise Res as an empty vector of ground atoms;
3:     ≻^⊗ ← ⊗(≻_M, ≻_P); // One of the Algorithms 1 to 4
4:     return iteratedSkyline(KB, Q, ≻^⊗, k). // Algorithm 5
```

The following theorem proves the correctness of the k-Rank algorithm, and shows that it runs in polynomial time under certain conditions.

Theorem 3.3.5. Let $KB = (O, P, M, \otimes)$, with $O = (D, \Sigma)$, be a PP_Datalog$^{\pm}$ ontology, Q be a DAQ, and $k \geq 0$. Then,

(i) Algorithm k-Rank correctly computes a k-rank answer to Q over KB; and

(ii) if O is a guarded Datalog$^{\pm}$ ontology, and $\otimes$ can be computed in $O(poly(D) \cdot S)$, then the running time of k-Rank is $O(poly(D) \cdot S)$, where S is the cost of computing $\Pr_M(a)$ for any atom a such that $O \models a$.

Proof sketch. (i) Correctness is a consequence of the direct application of the definition of k-rank: the while loop in line 5 iteratively computes the skyline answers to Q by means of a subroutine, adds these results to the output in arbitrary order, and removes them from consideration. Line 10 ensures that at most k results are returned.

(ii) Since O is guarded, answers to Q can be computed in polynomial time in the data complexity [31]. The $\otimes$ operator is computed in time $O(|\succ_P| \cdot S)$, and computeSkyline can be computed in polynomial time by a linear-time scan of the chase structure for Q (of polynomial size by hypothesis), and the results can be removed by another scan. □

As a consequence of Theorem 3.3.5 and Corollaries 3.3.2 and 3.3.4, one can conclude that if probabilities in M can be computed or approximated in polynomial time (in the data complexity), then so can k-rank answers.

3.4 Implementation and Evaluations

In the case of PP_Datalog$^{\pm}$, I have evaluated and analysed the running time of my algorithms over a combination of real-world and synthetic data. I use the IMDB for the probabilistic model and the semantic data, while the user preferences were randomly generated using a density factor. The code and dataset are available as open source [156].

This section evaluates and analyses the running time of my algorithms over a

combination of real-world and synthetic data. It first describes the experimental setup, and then continues to discuss the obtained results.

3.4.1 Implementation and Hardware

The framework was implemented by extending the Datalog$^{\pm}$ query answering engine in [77], which supports FO-rewritable fragments of the Datalog$^{\pm}$ family of ontology languages. All graph operations were implemented using the JGraphT[1] library, which provides efficient data structures for the representation of graph structures, as well as efficient implementations of operations such as reachability, transitive closure, and cycle detection. The whole code was written in Java

All runs were done on a laptop with an Intel Core i5 processor at 2.6 GHz and 16GB RAM, under the Windows 7 Professional (64-bit) SP1 operating system (Build 7601) and a Sun JVM Standard Edition with maximum heap size set to 14 GB RAM. To minimise experimental variation, all results are averages of between 5 and 10 independent runs.

3.4.2 Experimental Setup

Inputs to the system consist of tuples of the form $\langle Q, O(D, \Sigma), P, M, k\rangle$, where Q is the query, D is the database, Σ is a set of TGDs, P is the preference model, M is the probabilistic model and, k is the number of query results to be sorted according to the user's preferences. The preference graph is a labelled directed graph (N, E), where N is the node set (the atoms), and E is the edge set (the preference relation). The preference-augmented chase is then used for obtaining the answer to the query.

- *Data*: All runs were carried out using the ontology built on the basis of the IMDB dataset, as described in Section 3.4.4. To test my algorithms on instances of different sizes, I considered different subsets of the dataset, varying this parameter from 1,000 to 13,000 nodes (movies). Finally, the database for this ontology was stored in a PostgreSQL 9.3 database.

[1] `http://jgrapht.org/`

- *User preferences*: User preference graphs (SPOs) were randomly generated by first creating a set of nodes of the required size and then adding a certain number of edges. For the latter, I used different values of a *density* parameter to set the size of E — this simply refers to the number of edges as a percentage of $|N| \cdot (|N| - 1)/2$ (the maximum possible number of edges in a DAG). For transitively closed graphs (see discussion below), I used a seed density manually tuned to obtain the target number of edges after the transitive closure. The averages and standard deviations for each set of independent runs yielding a data point are reported in Tables 3.4 and 3.5.

- *Probabilistic model*: As described in Section 3.4.4, the IMDB dataset provides a numerical rating ranging from 0 to 10; to obtain a probability, these values were normalised using a Gaussian distribution with population values of 0.635 (mean) and 0.126 (standard deviation).

- *Query*: The used query is $Q(X) = \exists T, Y\ \mathit{movie}(X, T, Y)$ for all runs. This query represents a user requesting a set of k movies sorted according to a combination of own preferences, and the probability that each movie is good (according to the reviews available on IMDB).

Finally, in contrast to the general analysis of the worst case running times, in these experiments the focus is on the special case of the application to computing top-k query answers. As such, we may resort to a simple (but in practice very effective) optimisation of the operators that involves not computing the transitive closure as the final step. Though this is necessary in theory to obtain an SPO as the result of the combination operation, the information added by the transitive closure is superfluous for Algorithm k-Rank.

3.4.3 Results of the Experiment

To test the performance of my algorithms, I carried out experiments varying several parameters, namely, (i) SPO size and (ii) the number of answers (k). As for (i), since

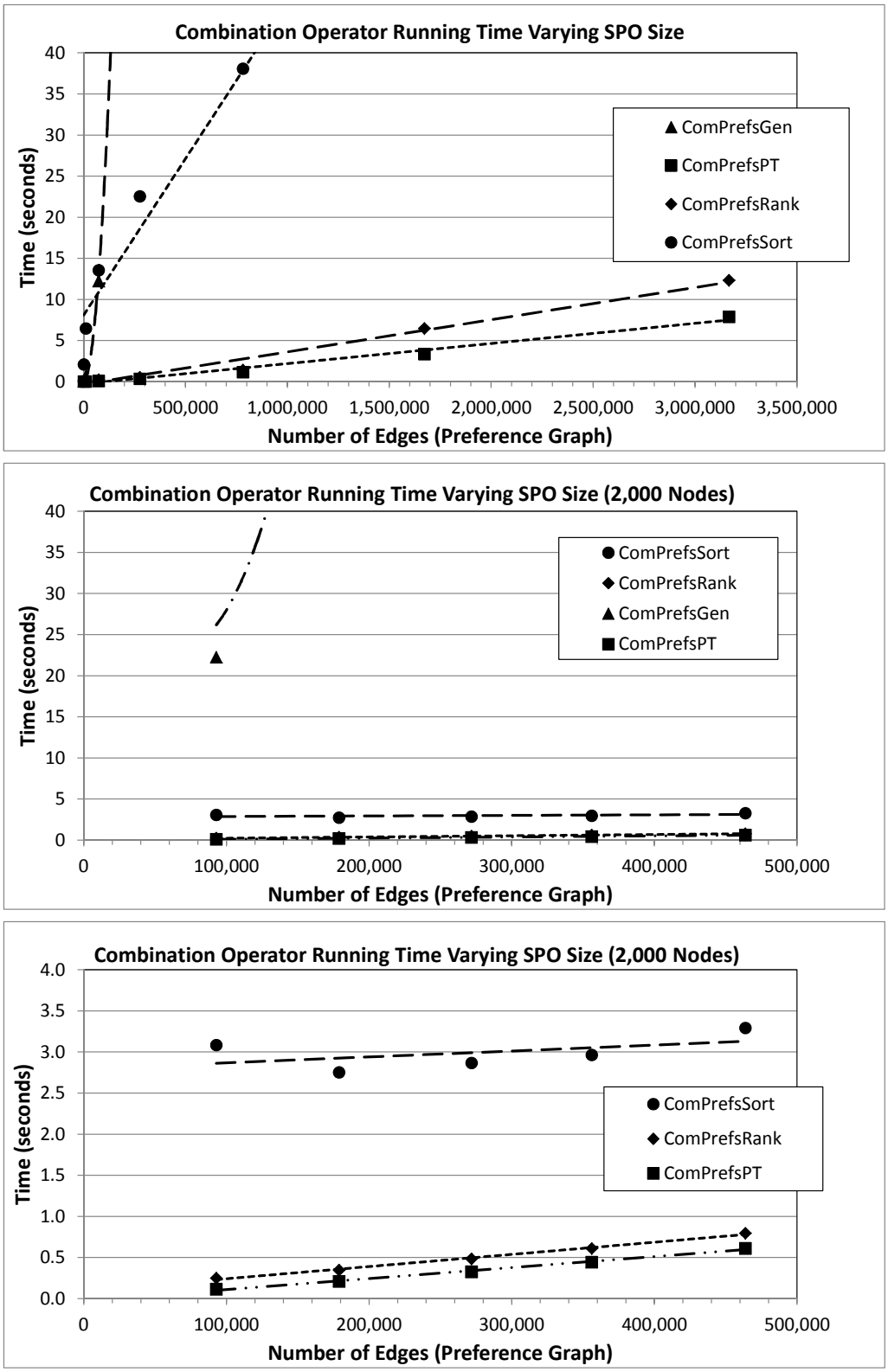


Figure 3.10: Experimental results over the IMDB dataset, and randomly generated SPOs. Each data series is augmented with the corresponding polynomial trend line with respect to the values in the x axis (cf. Section 3.1.2).

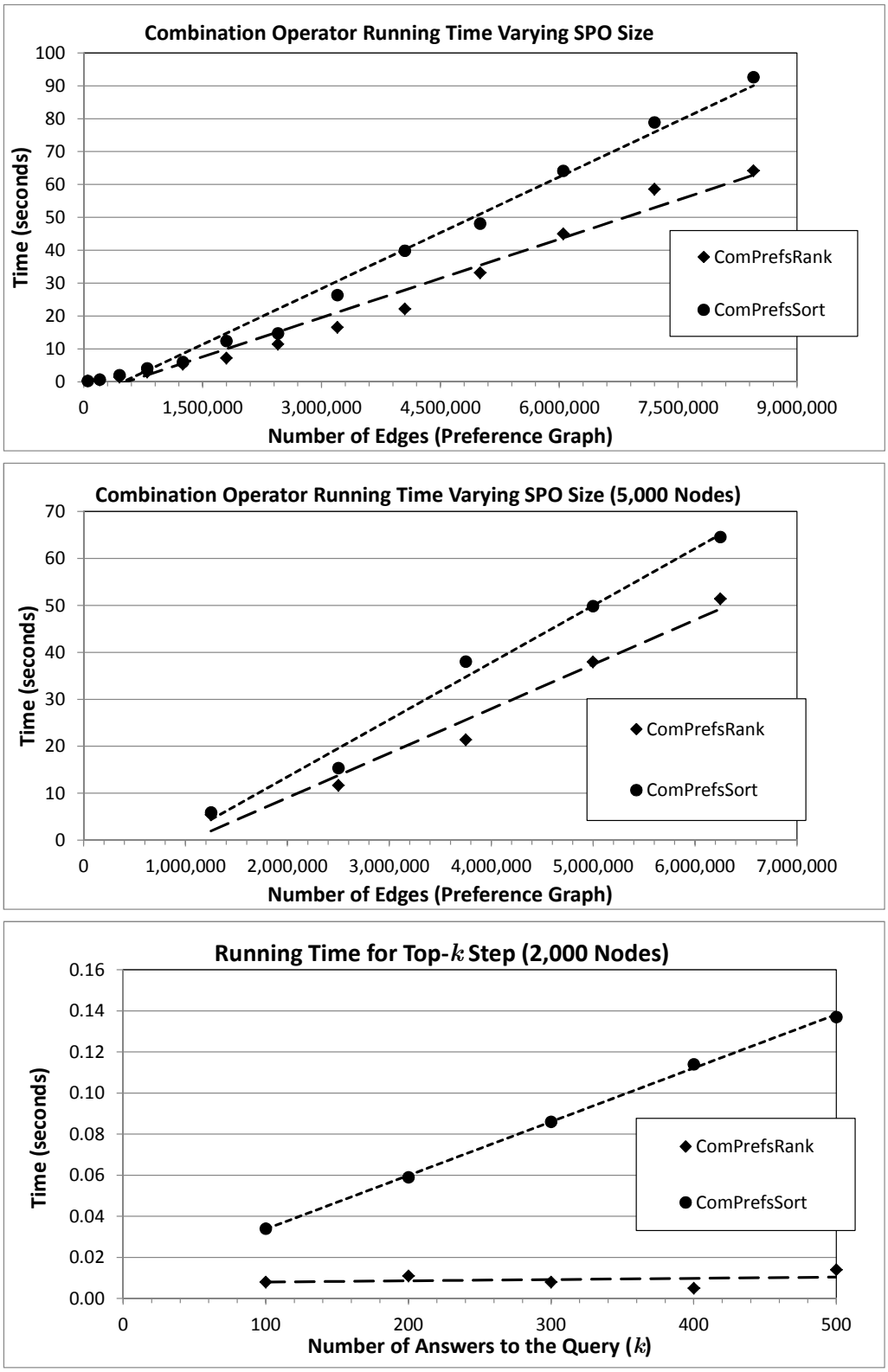


Figure 3.11: Experimental results over the IMDB dataset, and randomly generated SPOs. Each data series is augmented with the corresponding polynomial trend line with respect to the values in the x axis (cf. Section 3.1.2).

#Nodes	Avg. #Edges	St. Dev.	Avg. Density	Seed Density
1,000	940.8	60.401	0.19%	0.1%
2,000	10,520.8	943.89	0.53%	0.1%
3,000	73,728	6,091.8	1.64%	0.1%
4,000	275,681.2	10,503.84	3.45%	0.1%
5,000	781,634.8	38,602.54	6.25%	0.1%
6,000	1,672,068	35,328.19	9.29%	0.1%
7,000	3,167,134	84,153.84	12.93%	0.1%

Table 3.4: Information on the input graphs for Experiment 1.

the size of an SPO represented as a directed graph is given both by the number of nodes and the number of edges, my experiments show the effects of varying these two parameters.

Experiment 1: Varying the number of nodes in the SPO. Figure 3.10 (top) shows the running times of all four combination operators when varying the number of nodes from 1,000 to 13,000 with seed density of 0.1% (that determines the number of edges) and k fixed at 200 — the graph is plotted with number of edges in the x axis, since this parameter is the one that most directly affects the performance of the algorithms. Table 3.4 shows, for each data point, the number of nodes, average number of edges, standard deviation, average density, and seed density. Clearly, ComPrefsGen is greatly outperformed by the rest of the algorithms, while ComPrefsSort is the next worse. This is due to the fact that ComPrefsGen involves a pairwise comparison of all nodes (ground atoms) in the graph and, in each case, a check for potential cycles. On the other hand, ComPrefsSort suffers when the number of nodes increases, because it involves sorting each iterated skyline that, for these relatively low-density graphs, were quite large.

Experiment 2: Varying the edge density in the SPO. Next, the number of nodes is fixed in the SPO at 2,000, the density parameter changes from 0.2% to 0.4%, and k is fixed to 200; Table 3.5 shows the rest of the information for this experiment. The results can be seen in Figure 3.10 (middle). As in the previous

#Nodes	Avg. #Edges	St. Dev.	Avg. Density	Seed Density
2,000	92,918.9	7728.17	4.64%	0.20%
2,000	178,938.9	9917.13	8.95%	0.25%
2,000	271,800.7	13,082.45	13.60%	0.30%
2,000	356,077.3	15,205.12	17.81%	0.35%
2,000	463,820.2	15,216.49	23.20%	0.40%

Table 3.5: Information on the input graphs for Experiment 2.

#Nodes	#Edges	Density	Seed Density
1,000	49,950	10%	10%
2,000	199,900	10%	10%
3,000	449,850	10%	10%
4,000	799,800	10%	10%
5,000	1,249,750	10%	10%
6,000	1,799,700	10%	10%
7,000	2,449,650	10%	10%
8,000	3,199,600	10%	10%
9,000	4,049,550	10%	10%
10,000	4,999,500	10%	10%
11,000	6,049,450	10%	10%
12,000	7,199,400	10%	10%
13,000	8,449,350	10%	10%

Table 3.6: Information on the input graphs for Experiment 3 (a).

experiment, one can see that ComPrefsGen is outperformed by the rest; however, in this case, ComPrefsSort performs comparably to ComPrefsRank and ComPrefsPT, since the number of nodes to sort is fixed in this case. Figure 3.10 (bottom) shows the comparison without ComPrefsGen to illustrate the differences among the other three operators more clearly.

Experiment 3: Varying the number of nodes in the SPO (high edge density). (a) Figure 3.11 (top) corresponds to the same setup of the Experiment 1 with one important difference: the SPOs generated as inputs to the operators are not transitively closed. Though, by definition, this is required, algorithms ComPrefsRank and ComPrefsSort do not actually use this information — it is highly redundant and greatly hinders the performance of the algorithms. In this plot, one can see that

#Nodes	#Edges	Density	Seed Density
5,000	1,249,750	10%	10%
5,000	2,499,500	20%	20%
5,000	3,749,250	30%	30%
5,000	4,999,000	40%	40%
5,000	6,248,750	50%	50%

Table 3.7: Information on the input graphs for Experiment 3 (b).

these two operators remain quite scalable even for graphs with over 8M edges, which represents a much larger graph, if it were transitively closed.

(b) As before, I also ran another experiment in which the number of nodes was set to 5,000, and the density varied from 10% to 50% (also without transitive closure) — the results are shown in Figure 3.11 (middle).

Tables 3.6 and 3.7 show, for each data point, the number of nodes, number of edges, density, and seed density.

Experiment 4: Varying k. Figure 3.11 (bottom) shows the running time for the computation of the top-k answers without taking into account the combination operation; the number of nodes was fixed at 2,000 and density at 20%. For this experiment, I only show ComPrefsRank and ComPrefsSort, since this operation is mostly affected by the density of edges in the SPO. If the graph is too sparse, the first skyline usually contains all top-k answers — in the runs corresponding to the previous lower-density experiments, the running time for the top-k computation was negligible.

3.4.4 A Use Case

Let us now describe a real-world application of my formalism.

Data and Probabilistic Model. The raw data was obtained from the Internet Movie Database (IMDB)[2]; the resulting database consists of 13,893 movies — the structure of the used ontology is described in Figure 3.12. Furthermore, the IMDB

[2]www.imdb.com

Movie Ontology Predicates:

$movie(I, T, Y)$: Denotes a movie with identifier I, title T, and year of release Y — same schema for predicates corresponding to categories such as "action".
$actor(I, F, L, G)$: Denotes an actor with identifier I, first name F, last name L, and gender G.
$director(I, F, L)$: Denotes a director with identifier I, first name F, and last name L.

Tuple-generating Dependencies:

$$
\begin{aligned}
\Sigma_T = \{ & action(I,T,Y) \rightarrow movie(I,T,Y), & & adventure(I,T,Y) \rightarrow movie(I,T,Y),\\
& animation(I,T,Y) \rightarrow movie(I,T,Y), & & biography(I,T,Y) \rightarrow movie(I,T,Y),\\
& comedy(I,T,Y) \rightarrow movie(I,T,Y), & & crime(I,T,Y) \rightarrow movie(I,T,Y),\\
& documentary(I,T,Y) \rightarrow movie(I,T,Y), & & drama(I,T,Y) \rightarrow movie(I,T,Y),\\
& family(I,T,Y) \rightarrow movie(I,T,Y), & & fantasy(I,T,Y) \rightarrow movie(I,T,Y),\\
& film\_noir(I,T,Y) \rightarrow movie(I,T,Y), & & history(I,T,Y) \rightarrow movie(I,T,Y),\\
& horror(I,T,Y) \rightarrow movie(I,T,Y), & & music(I,T,Y) \rightarrow movie(I,T,Y),\\
& musical(I,T,Y) \rightarrow movie(I,T,Y), & & mystery(I,T,Y) \rightarrow movie(I,T,Y),\\
& romance(I,T,Y) \rightarrow movie(I,T,Y), & & sci\_fi(I,T,Y) \rightarrow movie(I,T,Y),\\
& sport(I,T,Y) \rightarrow movie(I,T,Y), & & thriller(I,T,Y) \rightarrow movie(I,T,Y),\\
& war(I,T,Y) \rightarrow movie(I,T,Y), & & western(I,T,Y) \rightarrow movie(I,T,Y),\\
& movie(M,T,Y) \rightarrow \exists D\, hasDirector(M,D), & & movie(M,T,Y) \rightarrow \exists A\, hasActor(M,A),\\
& actor(A,F,L,G) \rightarrow \exists M\, hasActor(M,A), & & director(D,F,L) \rightarrow \exists M\, hasActor(M,D)\}.
\end{aligned}
$$

Figure 3.12: Description of the movie ontology used in the use case and experimental evaluation.

associates a rating with each movie (computed as the average opinion of the users who give their input). Table 3.8 shows a small subset of the IMBD, which was selected for this use case. The ratings range from 0 to 1; probability values, which adequately model the opinions of the users about how good the movie is, were computed as the cumulative density function of a Gaussian distribution with mean 0.635 and standard deviation 0.126 (the population values corresponding to the entire database of 13,893 movies).

User Preferences. In the following, suppose a user has declared the following set of preferences, which I express as an ordered set of preference formulas (very similar to the approach introduced in [39] for preferences in relational databases). If a_1 and a_2 are atoms, a *preference formula pf* is of the form "$a_1 \succ a_2$ *if* $C(a_1, a_2)$", where $C(a_1, a_2)$ is a first-order formula. We call $C(a_1, a_2)$ the *condition* of *pf*, denoted *cond*(*pf*). Suppose the following preference formulas have been acquired for a given user:

$$
\begin{aligned}
pf_1: \quad & movie(M_1,T_1,Y_1) \succ movie(M_2,T_2,Y_2) \text{ if } drama(M_1,T_1,Y_1) \wedge drama(M_2,T_2,\\
& Y_2) \wedge hasActor(M_2,A_2) \wedge actor(A_2, brendan, fraser) \wedge \neg hasActor(M_1,A_2).
\end{aligned}
$$

Drama movies without Brendan Fraser are preferred over those with him

ID Year	Title Genre	Actors Rating	Director Probabilities
m_1	X-Men: First Class	M. Fassbender, J. McAvoy	B. Singer
2011	Action/Sci-fi	0.80	0.90
m_2	Haywire	E. McGregor, M. Fassbender	S. Soderbergh
2011	Action/Thriller	0.60	0.39
m_3	Gravity	G. Clooney, S. Bullock	A. Cuarón
2013	Action/Drama	0.85	0.96
m_4	Batman & Robin	G. Clooney, A. Schwarzenegger	J. Schumacher
1997	Action/Fantasy	0.40	0.03
m_5	The Prestige	S. Johansson, H. Jackman	C. Nolan
2006	Mystery	0.80	0.90
m_6	Crash	S. Bullock, B. Fraser	P. Haggis
2004	Drama	0.60	0.39
m_7	Elysium	M. Damon, J. Foster	N. Blomkamp
2013	Sci-fi/Action	0.68	0.64
m_8	Savages	B. Lively, T. Kitsch	O. Stone
2012	Adventure/Crime	0.70	0.69
m_9	2001: A Space Odyssey	K. Dullea, G. Lockwood	S. Kubrick
1968	Sci-fi	0.80	0.90
m_{10}	Eyes Wide Shut	T. Cruise, N. Kidman	S. Kubrick
1999	Drama	0.70	0.69
m_{11}	A Clockwork Orange	M. MacDowell, P. Magee	S. Kubrick
1971	Sci-fi	0.90	0.98
m_{12}	Ghost Busters II	B. Murray, D. Aykroyd	I. Reitman
1989	Action/Sci-fi	0.60	0.39
m_{13}	Forrest Gump	T. Hanks, R. Wright	R. Zemeckis
1994	Drama/Romance	0.90	0.98
m_{14}	Dragnet	T. Hanks, D. Aykroyd	T. Mankiewicz
1987	Comedy	0.60	0.39
m_{15}	Inception	L. DiCaprio, J. Gordon-Levitt	C. Nolan
2010	Action/Sci-fi	0.90	0.98

Table 3.8: Small subset of movies taken from the database to illustrate the use case.

pf_2 : $movie(M_1, T_1, Y_1) \succ movie(M_2, T_2, Y_2)$ *if* $(hasActor(M_1, A_1) \wedge actor(A_1,$ $matt, damon) \wedge hasActor(M_1, A_2) \wedge actor(A_2, george, clooney)) \wedge$ $\neg(hasActor(M_2, A_1) \vee hasActor(M_2, A_2))$.

Movies with George Clooney or Matt Damon are preferred over the movies without them

pf_3 : $movie(M_1, T_1, Y_1) \succ movie(M_2, T_2, Y_2)$ *if* $hasActor(M_1, A) \wedge hasActor(A, michael, fassbender) \wedge \neg hasActor(M_2, A)$.

Movies starring Michael Fassbender are preferred over all others

pf_4 : $movie(M_1, T_1, Y_1) \succ movie(M_2, T_2, Y_2)$ *if* $hasActor(M_1, A_1) \wedge hasActor(A_1,$ $scarlett, johansson) \wedge hasActor(M_2, A_2) \wedge hasActor(A_2, leonardo, dicaprio)$.

Movies with Scarlett Johansson are preferred over those with Leonardo DiCaprio.

pf_5 : $movie(M_1, T_1, Y_1) \succ movie(M_2, T_2, Y_2)$ *if* $(action(M_1, T_1, Y_1) \wedge (sci\_fi(M_1, T_1, Y_1)) \wedge drama(M_2, T_2, Y_2))$.

Action and science fiction movies are preferred over dramas

pf_6 : $movie(M_1, T_1, Y_1) \succ movie(M_2, T_2, Y_2)$ *if* $sci\_fi(M_1, T_1, Y_1) \wedge sci\_fi(M_2, T_2, Y_2) \wedge hasDirector(M_1, D_1) \wedge director(D_1, stanley, kubrick) \wedge hasDirector(M_2, D_2) \wedge director(D_2, ivan, reitman)$.

Science fiction movies directed by Stanley Kubrick are preferred over those that were directed by Ivan Reitman

pf_7 : $movie(M_1, T_1, Y_1) \succ movie(M_2, T_2, Y_2)$ *if* $hasDirector(M_1, D_1) \wedge director(D_1, stanley, kubrick) \wedge hasDirector(M_2, D_2) \wedge director(D_2, oliver, stone)$.

Movies directed by Stanley Kubrick are preferred over those by Oliver Stone.

To avoid cyclic preferences, these preference formulas are applied in the given order, and only apply if no cycles are introduced. Figure 3.13 shows the induced preference relation over the set of movies in Table 3.8.

Results. To illustrate the different behaviours of the preference merging operators of Section 3.1.2, I study the answers to the query $Q(X) = \exists T, Y\ movie(X, T, Y)$ obtained via Algorithm ComPrefsRank with each one — the results are shown in Table 3.9. For each rank i from 1 to 13, the table shows the set of IDs corresponding to movies (cf. Table 3.8) that have rank i in the graph resulting from merging the original user SPO with the one induced by the probabilistic model using several combinations operator and parameter values. For example, using the merging operator from Algorithm 2 with $f = min$, the table shows that movies m_1 and m_2 have rank 2 in the combined SPO. Therefore, a k-answer is built by making an arbitrary choice from the set for each rank from 1 to k. For example, one possible 5-rank answer using Algorithm 2 with $f = min$ is $\langle m_4, m_2, m_5, m_{10}, m_6 \rangle$.

This table shows interesting differences in the results obtained using each merging operator with different parameters. For instance, movie m_3 receives rank 1 in all cases, since it is one of the user's favourites, and it also has a very high probability

of 0.96. However, note that for m_4, which is another of the user's favourites, the behaviour of each algorithm is quite different.

Algorithm 1 (with both $t = 0.15$ and $t = 0.30$) assigns rank 3 to this movie, which is a compromise between the fact that the user ranked it very high, but its probability is almost zero.

On the other hand, *Algorithm 2* has two variants depending on the input function f: with *min*, the algorithm makes optimistic choices and ranks high movies that are either ranked high by the user or have high probability — in this case, m_4 is assigned the top rank. With $f = max$, the algorithm behaves pessimistically by assigning the worst rank possible between the user and the probability — since m_4 is the lowest ranking movie relative to probability, it receives the worst rank in the result.

Algorithm 3 uses the probability values to purge the user's SPO, using as a result the structure arising from the deletion of the movies that do not surpass the threshold. Note that movie m_4 thus disappears from all possible answers. Another interesting observation is that with $p = 0.95$, only the highest probability choices are kept — however, the user's preferences are still visible in the resulting structure. For instance, movie m_{15} climbed a position to join m_{11}, because the ones dominating it (m_9 and m_5) were removed; however, m_{13} is still less preferable than m_{15} and m_{11}.

Finally, the behaviour of *Algorithm 4* is quite clear: it uses the probability values to sort the iterated skylines of the user's preferences — the only non-singleton sets in the rank assignments thus only arise when ties occur in probability values. This kind of answer is perhaps most useful when the user desires to have a fully ordered list of choices; on the other hand, the other algorithms may be most applicable to the presentation of answers in a layered fashion, such as an online system that shows movie suggestions by page.

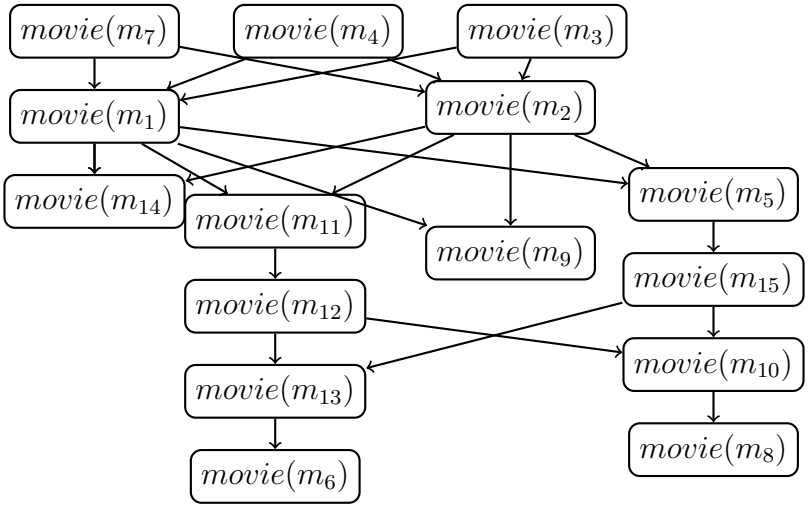


Figure 3.13: The user's SPO modelled by preference formulas pf_1 to pf_7.

3.5 Summary

This chapter presented an extension of the Datalog$^{\pm}$ family of ontology languages for preference-based query answering under uncertainty. The main focus of this work has been on defining preference combination operators, which produce a preference relation, given a general SPO and a score-based preference relation. I analysed four specific algorithms for such an operator, and investigated their semantic and computational properties. Finally, I have studied a basic algorithm for answering k-rank queries, and showed that under certain conditions, k-rank queries can be answered in polynomial time in the data complexity, which is the same complexity as answering traditional preference-based queries in relational databases. I have also reported on an implementation and experimental results. In the next chapter, I will extend this formalism, to the case where we have the preferences of a group of users, together with uncertainty.

Rank	**Alg. 1** $t = 0.15$	**Alg. 1** $t = 0.3$	**Alg. 2** $f = min$	**Alg. 2** $f = max$	**Alg. 3** $p = 0.4$	**Alg. 3** $p = 0.95$	**Alg. 4**
1	$\{m_3\}$	$\{m_3, m_7\}$	$\{m_3, m_4, m_7, m_7, m_{11}, m_{13}, m_{15}\}$	$\{m_3\}$	$\{m_7, m_3\}$	$\{m_3\}$	$\{m_3\}$
2	$\{m_1\}$	$\{m_1\}$	$\{m_1, m_2\}$	$\{m_1, m_5, m_9, m_{11}\}$	$\{m_1\}$	$\{m_{11}, m_{15}\}$	$\{m_7\}$
3	$\{m_7, m_4\}$	$\{m_4\}$	$\{m_5, m_{14}, m_9\}$	$\{m_{15}\}$	$\{m_5, m_9, m_{11}\}$	$\{m_{13}\}$	$\{m_4\}$
4	$\{m_9, m_{11}\}$	$\{m_9, m_{11}, m_5\}$	$\{m_8, m_{10}, m_{12}\}$	$\{m_7\ m_{10}, m_{13}\}$	$\{m_{15}\}$		$\{m_1\}$
5	$\{m_5\}$	$\{m_2\}$	$\{m_6\}$	$\{m_2, m_6, m_8, m_{12}, m_{14}\}$	$\{m_{13}, m_{10}\}$		$\{m_2\}$
6	$\{m_2\}$	$\{m_{14}, m_{15}\}$		$\{m_4\}$	$\{m_8\}$		$\{m_{11}\}$
7	$\{m_{14}, m_{15}\}$	$\{m_{13}\}$					$\{m_9, m_5\}$
8	$\{m_{10}, m_{13}\}$	$\{m_{12}\}$					$\{m_{14}, m_{15}\}$
9	$\{m_{12}\}$	$\{m_{10}, m_6\}$					$\{m_{12}\}$
10	$\{m_8, m_6\}$	$\{m_8\}$					$\{m_{13}\}$
11							$\{m_{10}\}$
12							$\{m_8\}$
13							$\{m_6\}$

Table 3.9: Answers to the query $Q(X) = \exists T, Y\ movie(X, T, Y)$ for several different combinations of merging operators and parameter values, using the set of movies shown in Table 3.8 and the SPO in Figure 3.13. The answers are shown according to their rank in the graph resulting from merging the original SPO with the probability-based preference relation.

Chapter 4

Group of Users Qualitative Preferences

In this chapter, I develop a novel integration of ontology languages with both preferences of groups of users and uncertainty management mechanisms.

This is a generalisation for a group of users of PP_Datalog$^{\pm}$ introduced in Chapter 3. I develop an extension of the Datalog$^{\pm}$ family of ontology languages [31] with a preference model over the consequences of the ontology, as well as a probabilistic model that assigns probabilities to them. The preference and the probabilistic model are assumed to model the preferences of a group of users and the uncertainty in the domain, respectively. I focus on answering k-rank queries in this context, presenting different strategies to compute group preferences as an aggregation of the preferences of a collection of single users. I then provide algorithms to answer k-rank queries for DAQs (disjunctions of atomic queries) under these group preferences and uncertainty that generalises top-k queries based on the iterative computation of classical skyline answers. I show that such DAQ answering in Datalog$^{\pm}$ can be done in polynomial time in the data complexity, under certain reasonable conditions, as long as query answering can also be done in polynomial time (in the data complexity) in the underlying classical ontology. Finally, I present a prototype implementation of

the query answering system, as well as experimental results (on the running time of my algorithms and the quality of their results) obtained from real-world ontological data and preference models, derived from information gathered from real users, showing in particular that my approach is feasible in practice.

The rest of this chapter is organised as follows. Section 4.1 introduces the language GPP_Datalog$^{\pm}$, its syntax and semantics, in particular, the general group preference model and the probabilistic model, along with preference merging and aggregation operations. Section 4.2 presents algorithms for query answering, along with correctness and data tractability results. Section 4.3 studies a set of properties that can be considered desirable for the aggregation of group preferences. The experimental setup and results are presented in Section 4.4. Finally, Section 4.5 summarises the main results of this chapter and gives an outlook on future research.

4.1 Formalism

I now introduce two ontology languages [117–120], one which combines the Datalog$^{\pm}$ ontology language with group preferences (a generalisation of preference handling in relational databases), called GP-Datalog$^{\pm}$, and one which combines Datalog$^{\pm}$ with both preferences of a group of users and probabilistic uncertainty, called GPP-Datalog$^{\pm}$. To my knowledge, these are the first combinations of ontology languages with group preferences with and without probabilistic uncertainty. I provide aggregation operators that take as input group preferences models, and compute an aggregated model from all individual preferences.

4.1.1 GPP_Datalog$^{\pm}$: Syntax and Semantics

In this section, I introduce the GPP_Datalog$^{\pm}$ language, an extension of Datalog$^{\pm}$ with both a group preference model and a probabilistic model; it is based on several previously proposed formalisms: the Pref-Datalog$^{\pm}$ language described above [115] (this language contemplates neither probabilities nor groups), PP_Datalog$^{\pm}$ [116] (which includes probabilities, but not groups), and GPref-Datalog$^{\pm}$ [117] (including

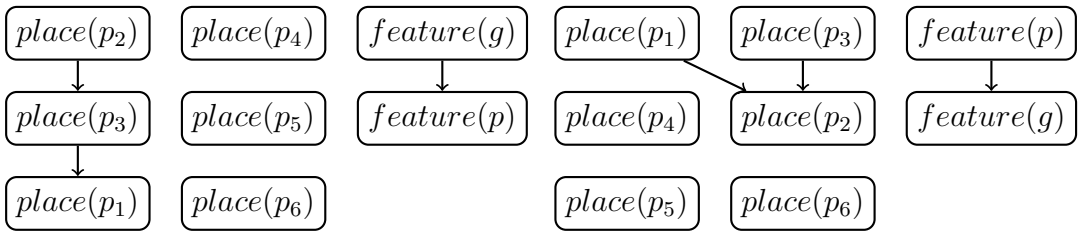


Figure 4.1: User u_1's preference relation. Figure 4.2: User u_2's preference relation.

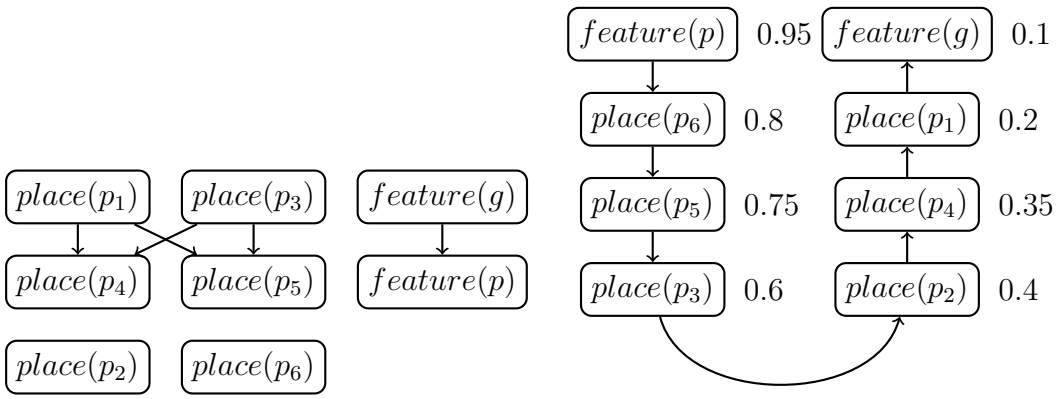


Figure 4.3: User u_3's preference relation. Figure 4.4: Probabilistic model $\succ_M$.

groups, but not probabilities). As discussed in this chapter, the combination of all three characteristics yields challenges not present in these previous proposals.

Group Preference Model. Let us start by generalising the preference models presented in Section 2.2.3 to represent preferences about groups of individuals. Specifically, a *group preference model* $\mathcal{U} = (U_1, \ldots, U_n)$ for $n \geq 1$ users is a collection of n user preference models.

Example 4.1.1. Consider again the Datalog$^{\pm}$ ontology $O = (D, \Sigma)$ from Example 2.1.1. Suppose that I have the following group preference model for three users $\mathcal{U} = (U_1, U_2, U_3)$:

$$U_1: \begin{cases} C_{1,1} : \mathit{place}(X) \succ \mathit{place}(Y) \text{ if } \mathit{propertyFeature}(X, p) \wedge \mathit{propertyFeature}(Y, g) \wedge X \neq Y, \\ C_{1,2} : \mathit{feature}(X) \succ \mathit{feature}(Y) \text{ if } \mathit{propertyFeature}(p_1, X) \wedge \mathit{propertyFeature}(p_2, Y) \wedge X \neq Y, \end{cases}$$

$$U_2: \begin{cases} C_{2,1} : \mathit{place}(X) \succ \mathit{place}(p_2) \text{ if } \mathit{property}(X, \mathit{apt}, l_1, e), \\ C_{2,2} : \mathit{feature}(X) \succ \mathit{feature}(g) \text{ if } \mathit{propertyFeature}(p_2, X) \end{cases}$$

$$U_3: \begin{cases} C_{3,1} : \mathit{place}(X) \succ \mathit{place}(Y) \text{ if } \mathit{property}(X, \mathit{apt}, l_1, C) \wedge \mathit{property}(Y, \mathit{apt}, l_2, D) \wedge X \neq Y, \\ C_{3,2} : \mathit{feature}(g) \succ \mathit{feature}(X) \text{ if } \mathit{propertyFeature}(p_2, X) \wedge X \neq Y. \end{cases}$$

Figures 4.1 to 4.3 (where I assume the transitive closure of the graphs) shows the SPOs induced by the preference models U_1, U_2, and U_3. Note that, for example, the preference over garage (*feature*(g)) and pool (*feature*(p)) are contradictory for the users of the group. ∎

Probabilistic Model. I assume the existence of a probabilistic model M that represents a probability distribution $\Pr_M$ over some set $X = \{X_1, \ldots, X_n\}$ of Boolean variables such that there is a 1-to-1 mapping from X to $\mathcal{H}_{Ont}$. Examples of the type of probabilistic models that I assume in this work are Markov logic and Bayesian networks. The probabilistic extension adopted here was first introduced in [74, 75].

In the rest of this chapter, I assume the existence of a preference relation $\succ_M$, defined as $a \succ_M b$ if $\Pr_M(a) > \Pr_M(b)$ — I refer to $\succ_M$ as the probability-based (preference) relation.

Example 4.1.2. Continuing with the running example, suppose that we have access to an online probabilistic model that assigns probabilities specifying how likely it is that certain events happen. This uncertainty could arise, for instance, from the fact that the system is aggregating information from multiple sources, which may contain conflicting information, as well as uncertainty due to other factors. Such a system could inform the user of the probability of a property, feature, of being available and recommendable at the time of the query by taking into account reviews, crowds, season, etc. For instance, we can query the probability of a particular place, or type of place, of being open with availability of space, or the probability of finding in the area at a certain time a specific feature.

Figure 4.4 gives an example of such a probability assignment, along with the preference relation as a graph that is induced by these values, assuming that higher probabilities are more preferable. The model M assigns to *feature*(p) the highest probability, while it assigns to *feature*(g) the lowest. ∎

GPP_Datalog$^{\pm}$ Ontologies. Let us now define GPP-Datalog$^{\pm}$ ontologies.

Definition 4.1.1. A *GPP-Datalog*$^{\pm}$ *ontology* has the form $KB = (O, \mathcal{U}, M)$, where O is a Datalog$^{\pm}$ ontology (with Herbrand base $\mathcal{H}_{Ont}$), $\mathcal{U} = (U_1, \ldots, U_n)$ is a group preference model with $n \geq 1$, and M is a probabilistic model.

4.2 Algorithms and Complexity

This section presents algorithms for answering k-rank queries for disjunctions of atomic queries, which generalise top-k queries based on the iterative computation of classical skyline answers. Moreover, it shows that answering disjunctions of atomic queries in GPP_Datalog$^{\pm}$ and *GP_Datalog*$^{\pm}$ is possible in polynomial time under certain conditions.

4.2.1 Query Answering in GPP_Datalog$^{\pm}$

In this section, I concentrate on skyline queries [16], a well-known class of queries that can be issued over preference-based formalisms — intuitively, answers to skyline queries consist of those elements that are not dominated by any other according to the input preferences. I also focus on the iterated computation of skyline answers [39] that allows us to assign a *rank* to each answer by iterating the following procedure: (1) initialise ℓ to 0; (2) assign the rank ℓ to the skyline answers; and (3) remove from consideration all answers of rank ℓ, increment ℓ by one, and go back to (2). As a generalisation of classical top-k queries, it adopts k-rank queries; answers to them are k-tuples of answers sorted by rank.

As seen in Figures 4.1 to 4.4, there are two challenges encountered in query answering with GPP_Datalog$^{\pm}$ ontologies. The first challenge is that each user preference model yields a certain precedence relation that might be in disagreement with the one induced by the probabilistic model. The second challenge is that the user preference models may be in disagreement with each other.

4.2.1.1 Preference Merging

To address the first challenge, I introduce the notion of *preference merging operators*, which are functions that take two preference relations, and produce a third one satisfying two basic properties.

Definition 3.1.2 (in Section 3.1) presents the two minimal properties required to produce a "reasonable" merging of the two relations.

For example, Algorithm ComPrefsGen (Algorithm 1) presents one such operator. Given a relation $\succ_U$, probability-based relation $\succ_M$, and value $t \in [0, 1]$ (that allows the user to choose how much influence the probabilistic model has on the output preference relation), the algorithm works by iterating through all pairs (a, b) of elements in $\succ_U$ and, if (i) $\succ_M$ disagrees with $\succ_U$, (ii) the difference in probability is greater than t, and (iii) changing (a, b) to (b, a) does not introduce a cycle in the associated graph, then the pair is inserted in reverse order into the output; otherwise, the output contains the same pair as $\succ_U$. Finally, the algorithm outputs the transitive closure of this relation. In the rest of this chapter, I use the notation $\otimes_t$ to denote the particular instance of this operator $\otimes$ given a specific value of t.

The following is an example of this merging operator.

Example 4.2.1. Consider again the running example. Figure 4.5 shows the result of the individual merging of the preference relation for each user with the probability-based relation using $t = 0$ for u_1, $t = 0.25$ for u_2, and $t = 0.4$ for u_3. Observe that even though user u_2 prefers *feature*(g) to *feature*(p), after merging with $\succ_M$, this preference is reversed (Figure 4.5). ∎

4.2.1.2 Preference Aggregation

To address the second challenge, I define *preference aggregation operators*, which are functions that take a set of preference models, and produce a new one.

Definition 4.2.1. Let $\mathcal{U} = (U_1, \ldots, U_n)$ be a group preference model, where every U_i is an SPO. A *preference aggregation operator* $\uplus$ on $\mathcal{U}$ yields an SPO $\succ^{\uplus}$.

A simple example of an aggregation operator is to consider all pairs of elements and do a Pareto composition [39].

Let us now explore two different approaches to a preference aggregation operator $\uplus$. In the first one, called *collapse to single user (CSU)*, one reduces the group modelling problem to a single-user problem by creating a single virtual user that is constructed by aggregating the preferences of the individuals from the group. In the second approach, called *voting-based aggregation*, one first computes the k-rankings according to each individual user, and then apply aggregation techniques based on voting strategies (originally developed for quantitative preferences [127]) to aggregate the relations induced by the rankings.

Collapse to Single User. Under the CSU strategy, the preference relation for all users, along with the probabilistic preference relation, are taken into account in the generation of a new preference relation that encodes the dominant preferences. This single-user preference relation is then used to compute the answers to queries.

The following algorithm computes this particular approach to defining a $\uplus$ operator; below, I provide an algorithm that uses this operator for answering k-rank queries to GPP_Datalog$^{\pm}$ ontologies in polynomial time in the data complexity (modulo the cost of computing probabilities with respect to the probabilistic model M).

Algorithm 7 (AggPrefsCSU) implements the preference aggregation operator $\uplus_{CSU}$; the output is a new preference relation consisting of the collapsed preferences of all relations. A graph is used as an intermediate data structure representing the collapsed preferences; the nodes of this graph are all the atoms that appear in the preference relations, while the edges are labelled with an integer representing the number of relations that have this edge in their individual graph. The algorithm iterates through all the users i, looks at all the edges in *currUserG*, and updates the general graph G by incrementing or decrementing the edge labels, and introducing or removing edges. After the final iteration of the for-loop in Line 3, the edge labels of

Algorithm 7 An algorithm for combining the relations in a group preference model with a probabilistic preference relation.

Input: SPOs $\succ_{U_1}, \ldots, \succ_{U_n}$.
Output: Preference relation $\succ^{\uplus} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.

```
1: function AggPrefsCSU(≻_{U_1}, ..., ≻_{U_n})
2:     Initialise G as an empty graph;
3:     Add as nodes in G all elements appearing in the preference relations ≻_{U_i};
4:     for all users U_i with i ∈ {1, ..., n} do
5:         Initialise currUserG as the graph corresponding to ≻_{U_i};
6:         for all edges (u, v) ∈ currUserG do
7:             if (u, v) ∉ G then // there is no edge (u, v) in G
8:                 Add edge (u, v) to G;
9:                 label_G(u, v) ← 1;
10:            if (u, v) ∈ G and label_G(u, v) ≥ 1 then // label of edge (u, v) ≥ 1
11:                label_G(u, v) ← label_G(u, v) + 1;
12:            if (v, u) ∈ G and label_G(v, u) = 1 then
13:                Remove edge (v, u) from G;
14:            if (v, u) ∈ G and label_G(v, u) > 1 then
15:                label_G(v, u) ← label_G(v, u) − 1;
16:    return getPreferenceRelation(removeCycles(transitiveClosure(G))).
```

G correspond to the number of users that have that edge in their preference relation. The final step of the algorithm computes the transitive closure of the graph, and eliminates any cycles by applying the procedure removeCycles (note that cycles can arise even though all individual relations are cycle-free). This subroutine does not unnecessarily remove edges, if there does not exist an edge e in $G \setminus \mathsf{removeCycles}(G)$ such that $\mathsf{removeCycles}(G) \cup \{e\}$ does not contain cycles. Since this subroutine is not explicitly given, $\uplus_{CSU}$ is actually a family of operators.

Example 4.2.2. Continuing from Example 4.2.1, Figure 4.6 shows the final collapsed graph. Consider the atoms *feature*(p) and *feature*(g) in this graph. As mentioned in Example 4.2.1, the preference of user u_2 was reversed, because of the probabilities for those items; as a result, we have an unanimous preference of *feature*(p) over *feature*(g) in the final graph. ∎

The following theorem states that $\uplus_{CSU}$ satisfies Definition 4.2.1.

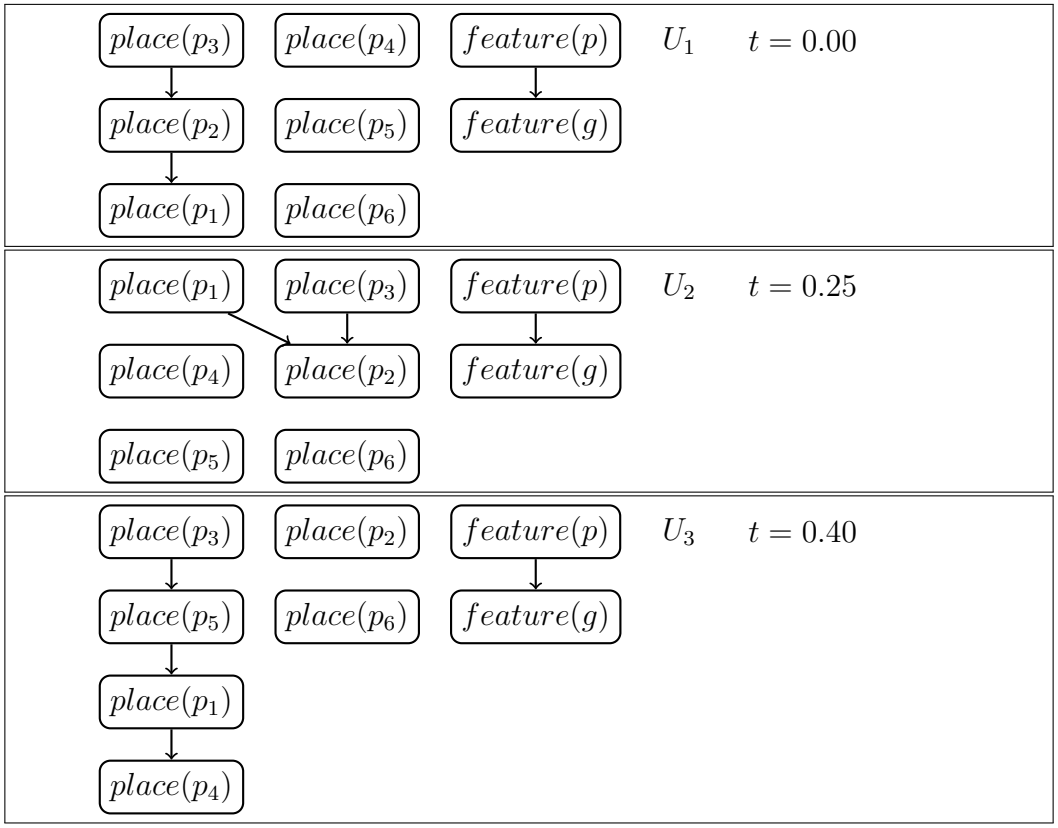


Figure 4.5: The merged preference relation obtained for each user. The three graphs show the merging of each individual preference with the probabilistic preference.

Theorem 4.2.1. Let $KB = (O, \mathcal{U}, M)$ be a GPP_Datalog$^{\pm}$ ontology, where $\mathcal{U} = (U_1, \ldots, U_n)$. Let $\succ^{\uplus} = \uplus_{CSU}(\succ_{U_1}, \ldots, \succ_{U_n})$. Then, the following properties hold: (i) if *removeCycles* preserves transitivity, then $\succ^{\uplus}$ is a preference aggregation operator; and (ii) if removeCycles only removes edges (v_1, v_2) whenever there does not exist another edge in the cycle labelled with a lower number, then $a_1 \succ^{\uplus} a_2$ for all a_1, $a_2 \in \mathcal{H}_{Ont}$ such that (a_1, a_2) in U_i for all U_i in $\mathcal{U}$.

Proof. (i) By hypothesis, $\succ^{\uplus}$ is transitive and cycle-free — the former is assumed by hypothesis and the latter is ensured by removeCycles. Therefore, $\succ^{\uplus}$ is irreflexive, and thus an SPO.

(ii) By Section 3.2, for $1 \leq i \leq n$, we know that $\otimes(M, \succ_{U_i})$ are all SPOs and therefore cycle-free. Towards a contradiction, suppose that $a_1 \not\succ^{\uplus} a_2$; by the construction of the graph G in AggPrefsCSU and the hypothesis that $(a_1, a_2) \in \otimes_{t_i}(M, \succ_{U_i})$ for all

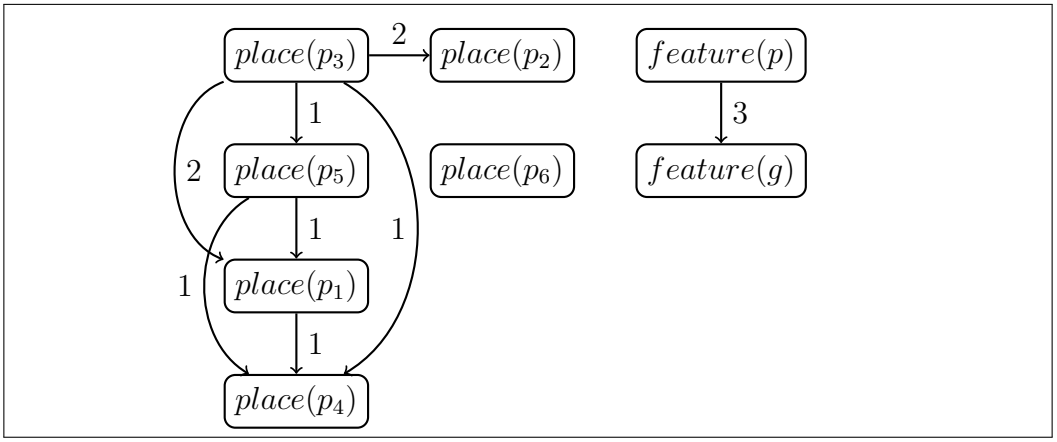


Figure 4.6: Collapse to single user graph.

$1 \leq i \leq n$, this can only happen if (a_1, a_2) belongs to a cycle, and the algorithm chose to remove this cycle by deleting this edge. Since the label of (a_1, a_2) is n and, by the assumption of how removeCycles works, the entire cycle is composed of edges labelled with n; but this contradicts $\otimes_{t_i}(M, \succ_{U_i})$ being SPOs. □

Voting-based Preference Aggregation. As an alternative to the approach described in the previous section, let us now briefly discuss specific strategies that can be used to combine the answers to k-rank queries computed individually for each user based on a small set of well-known voting mechanisms from the social choice literature. Recall that this is essentially different from the CSU approach above, where a single k-ranking is computed from a preference relation distilled from all the users' individual preferences. I consider the following voting mechanisms: *plurality voting*, where each user votes for their top-preferred items, the items' frequency for all the users are summed up, and the items with highest number of votes win; the *least misery* strategy first removes from consideration the elements that are the least preferred by each user, and then applies plurality voting — the idea behind it is that a group is as happy as its least happy member; in the *average without misery* strategy, the least misery approach is generalised by removing the t least liked elements for each member (instead of just one); and the *fairness strategy*, which is often applied when people try to fairly divide a set of items — one person chooses

first, then another, until everyone has made one choice, and next, everybody chooses a second item, often starting with the person who had to choose last in the previous round; an advantage of this strategy is that the top items from all individuals are always selected.

In the following, I use $\uplus_{vote}$ to denote the family of preference aggregation operators defined as: (i) for each input SPO, compute a ranking (linear order); and (ii) apply a voting-based strategy over these linear orders. For particular voting strategies, I use specific names such as $\uplus_{plu}$, $\uplus_{plu,fair}$, $\uplus_{plu,mis}$, and so on. Therefore, one must adapt the voting strategies to work over linearly sorted lists of elements. Plurality voting then works by assigning one vote to each element in the list. If a *misery* strategy is used, then each user's undesired elements are marked as unavailable before obtaining the individual rankings. Finally, if *fairness* is used, one iterates through each user, who picks their highest-ranked elements in turn.

Example 4.2.3. Consider the preference relations for users u_1, u_2, and u_3 from Example 4.2.1 from the running example. The first step towards applying a voting-based aggregation technique is to compute a ranking for each user from their corresponding SPOs. For $k = 4$, the following are some possible k-rankings for these users:

$$\begin{aligned} r_1 &= \langle \mathit{place}(p_3), \mathit{place}(p_4), \mathit{place}(p_5), \mathit{feature}(p) \rangle, \\ r_2 &= \langle \mathit{place}(p_1), \mathit{place}(p_3), \mathit{place}(p_5), \mathit{feature}(p) \rangle, \\ r_3 &= \langle \mathit{place}(p_3), \mathit{place}(p_2), \mathit{place}(p_6), \mathit{feature}(p) \rangle. \end{aligned}$$

If we apply the aggregation operator $\uplus_{plu}$, then we have three votes for $\mathit{place}(p_3)$, three votes for $\mathit{feature}(p)$, two votes for $\mathit{place}(p_5)$ and $\mathit{place}(p_6)$, one vote for $\mathit{place}(p_4)$, $\mathit{place}(p_1)$ and, $\mathit{place}(p_2)$, and no votes for $\mathit{feature}(g)$. The SPO output by $\uplus_{plu}$ arises from this tally. ∎

Algorithm 8 Algorithm for computing a k-rank answer to DAQ Q.

Input: (i) GPP_Datalog$^{\pm}$ $KB = (O, \mathcal{U}, M)$, where $\mathcal{U} = (U_1, \ldots, U_n)$,
(ii) DAQ $Q(\mathbf{X})$,
(iii) $k \geq 0$,
(iv) combinator operator $\otimes$, and
(v) aggregation operator $\uplus$.
Output: k-rank answer $\langle a_1, \ldots, a_{k'} \rangle$ to Q, with $k' \leq k$.
Calls: Algorithm 5.

```
1: function k-Rank(KB, Q, k, ⊗, ⊎)
2:     for all users U_i with i ∈ {1, ..., n} do
3:         ≻_i ← ⊗(M, ≻_{U_i});
4:     ≻^⊎ ← ⊎(≻_1, ..., ≻_n);
5:     return iteratedSkyline(KB, Q, ≻^⊎, k). // Algorithm 5
```

4.2.2 Answering k-Rank Queries

In this section, I first consider disjunctive atomic queries (DAQs) and then briefly discuss the extension to conjunctive queries (CQs). I show that the former can be answered in polynomial time, while answering the latter turns out to be Σ_2^p-complete.

I use the standard notions of substitutions and most general unifiers. More specifically, a *substitution* is a mapping from variables to variables or constants. Two sets S and T *unify* via a substitution θ, if $\theta S = \theta T$, where θA denotes the application of θ to all variables in all elements of A; here, θ is a *unifier*. A *most general unifier (mgu)* is a unifier θ such that for all other unifiers ω, there is a substitution σ such that $\omega = \sigma \circ \theta$.

I now formally define skyline and k-rank answers to DAQs in a GPP_Datalog$^{\pm}$ ontology.

Definition 4.2.2. Let $KB = (O, \mathcal{U}, M)$ be a GPP_Datalog$^{\pm}$ ontology, where $\mathcal{U} = (U_1, \ldots, U_n)$, $\otimes$ and $\uplus$ be preference merging and combination operators, respectively, and let $Q(\mathbf{X}) = q_1(\mathbf{X}) \vee \cdots \vee q_n(\mathbf{X})$ be a DAQ. A *skyline answer* to $Q(\mathbf{X})$ relative to $\succ^{\uplus} = \uplus(\otimes(M, \succ_{U_1}), \ldots, \otimes(M, \succ_{U_n}))$ is any θq_i entailed by O such that no θ' exists with $O \models \theta' q_j$ and $\theta' q_j \succ^{\uplus} \theta q_i$, where θ and θ' are ground substitutions for the variables in $Q(\mathbf{X})$. For transitive relations, a *k-rank* answer to $Q(\mathbf{X})$, $k \geq 0$, is a

sequence $S = \langle \theta_1 q_{l_1}, \ldots, \theta_{k'} q_{l_{k'}} \rangle$ of maximal length of ground instances $\theta_i q_{l_i}$ of atoms q_{l_i} in $Q(\mathbf{X})$, built by subsequently appending the skyline answers to $Q(\mathbf{X})$, removing these atoms from consideration, and repeating the process until (a) either the length of S is k, or (b) no more skyline answers to $Q(\mathbf{X})$ remain.

In Definition 4.2.2, note that k-rank answers are only defined when the preference relation is transitive; this kind of answer can be seen as a generalisation of traditional top-k answers [151] that are still defined when $\succ^{\uplus}$ is not a weak order, and their name arises from the concept of *rank* introduced in [39]. Conceivably, other ways of sorting answers given an SPO are possible; here, I focus on iterated skylines, since it is the most general approach.

Intuitively, for DAQs, both kinds of answers can be seen as atomic consequences of O that satisfy the query: the skyline answers can be seen as sets of atoms that are not dominated by any other such atom, while k-rank answers are k-tuples sorted according to the preference relation. I refer to these as answers in *atom form*.

I now present a general algorithm for obtaining k-rank answers using the machinery developed up to now. Algorithm 8 presents the pseudocode — essentially, the algorithm follows Definition 4.2.2 by first applying the merging operator $\otimes$ to each user's preference relation and then applying the aggregation operator $\uplus$. Finally, the result is obtained by computing a k-rank over this aggregation.

Example 4.2.4. Consider the running example, with $Q = place(X)$, For $k = 3$ and $\mathbf{t} = (t_1,\ t_2,\ t_3) = (0, 0.1, 0.15)$, one possible k-rank answer to Q given by both k-$\mathsf{Rank}(KB, Q, k, \mathbf{t},\ \otimes_t, \uplus_{CSU})$ and k-$\mathsf{Rank}(KB, Q, k, \mathbf{t}, \otimes_t, \uplus_{plu})$ is $\langle place(p_3),\ place(p_6),\ place(p_2),\ place(p_5) \rangle$ —clearly, if different rankings from the ones computed in Example 4.2.3 are used, then the result with respect to $\uplus_{plu}$ could be different from that of $\uplus_{CSU}$. In this example, I use the following operator $\otimes_t = \mathsf{ComPrefsGen}(KB, t)$. ∎

The following theorem proves the correctness of the k-Rank algorithm, and it shows that it runs in polynomial time under certain conditions.

Theorem 4.2.2. Let $KB = (O, \mathcal{U}, M)$ be a GPP_Datalog$^{\pm}$ ontology, $\otimes$ and $\uplus$ be merging and aggregation operators, respectively, Q be a DAQ, and $k \geq 0$. Then, (i) Algorithm k-Rank correctly computes k-rank answers to Q; (ii) if $\uplus = \uplus_{CSU}$ and the removeCycles subroutine does not unnecessarily remove any edges, then k-Rank runs in $O(poly(\|D\|) \cdot S + C)$ time in the data complexity; and (iii) if $\uplus = \uplus_{vote}$, and $\uplus$ can be computed in polynomial time in the data complexity, then k-Rank runs in $O(poly(\|D\|) \cdot S)$ time in the data complexity. Here, $\|D\|$ denotes the size of D, $poly(\|D\|)$ is a polynomial in $\|D\|$, S is the cost of computing $\Pr_M(a)$ for any atom a such that $O \models a$, and C is the cost of removeCycles.

Proof.

(i) Immediate by the direct application of the definition of k-rank answers in k-Rank. (ii) As O is assumed to be tractable, the necessary finite portion of the chase of O relative to Q can be computed in polynomial time in the data complexity [31]. Every $\succ_i$ can be computed in time $O(\|\succ_{U_i}\| \cdot S)$, where $\|\succ_{U_i}\|$ is a polynomial in $\|D\|$. Hence, $\succ^{\uplus} = \uplus_{CSU}(\succ_1, \ldots, \succ_n)$ is computable in time $O(poly(\|D\|) \cdot S + C)$. Now, each of the k skyline calculations in iteratedSkyline$(KB, Q, \succ^{\uplus}, k)$ can be done in polynomial time by a linear-time scan of the chase structure for Q (of polynomial size, by hypothesis), and the results can be removed by another such scan.

(iii) Immediate by the proof of (ii), using the hypothesis that $\uplus$ can be computed in polynomial time in the data complexity. □

Note that the running time depends on the cost of the removeCycles subroutine. Though cycles can be removed in polynomial time, depending on the properties that we wish the output of this subroutine to satisfy, the actual cost may vary considerably — for instance, if we wish to remove the minimum number of edges possible, this is already NP-complete. The computational cost also depends on the implementation of the voting strategies, which can clearly be computed in polynomial time in the data complexity for the strategies discussed above; since cycles can never arise, the C factor from part (ii) does not appear.

k-Ranking for Conjunctive Queries. For (non-atomic) conjunctive queries, the substitutions in answers no longer yield single atoms but rather *sets* of atoms; thus, to answer such queries relative to a preference relation, we must extend the preference specification framework to take into account sets of atoms instead of individual ones. One such approach was proposed in [180], where a mechanism to define a preference relation over tuple sets $\succ_{PS}$: $2^{\mathcal{H}_{Ont}} \times 2^{\mathcal{H}_{Ont}}$ is introduced. In the following, I briefly treat their complexity and discuss how methods from the relational databases can be applied to answer them. The following result says that this change has a direct impact on the complexity of both skyline and k-rank query answering; its proof is a straightforward generalisation of the result presented in [116].

Theorem 4.2.3. Let $KB = (O, \mathcal{U}, M)$ be a GPP_Datalog$^{\pm}$ ontology, with $O = (D, \Sigma)$ and $\mathcal{U} = (U_1, \ldots, U_n)$. Let each $\succ_{U_i}$ describe a preference relation $\succ_{PS}$: $2^{\mathcal{H}_{Ont}} \times 2^{\mathcal{H}_{Ont}}$ such that membership can be tested in polynomial time. Let $\otimes$ and $\uplus$ be preference merging and aggregation operators, respectively, and let Q be a CQ. If the relational schema and Σ are fixed, deciding whether the set of skyline answers or a k-rank answer to Q are non-empty is Σ_2^p-complete.

The heuristic algorithms presented in [180] for relational databases can be applied to GPP_Datalog$^{\pm}$ by computing the *chase* relative to the query and thus materialising the necessary part of the ontology into a database. The result in [116] provides a way to leverage tools developed for relational databases for first-order (FO) rewritable fragments; if the user preferences are given as sets of preference formulas [39], this result can be easily extended for GPP_Datalog$^{\pm}$ ontologies.

4.3 Semantic Properties

This section presents several ways to compute group preferences as an aggregation of sets of single-user preferences, based on social choice theory.

Let us now study a set of properties that can be considered desirable for the results of the aggregation of group preferences. These properties are based on the ones

usually studied in social choice theory [69], but extended for k-rank query answers. Since the two kinds of aggregation strategies being studied are different in nature, I study a different set of properties for each; essentially, the ones for voting-based strategies focus on the ordering of elements (by individual users, subgroups, and groups), while the properties for CSU focus on the relationship between specific pairs of elements. Note that these properties are only with respect to the aggregation strategies, and do not involve the merging operations used to combine SPOs with preferences arising from probabilistic models.

In the presentation of the properties, when it says that *an element is added*, I refer to the modification of individual SPOs by adding a new element along with any preference edges between that element and existing ones; I assume that the rest of the SPO is left unchanged, except for the addition of any necessary edges to satisfy transitivity. Similarly, when I say that *an element is removed*, I refer to the operation of removing the element from the SPO, and any preference edges associated with it. In the sequel, let $\mathcal{U} = (U_1, \ldots, U_n)$, $\mathcal{U}_A = (U_1^A, \ldots, U_n^A)$, and $\mathcal{U}_B = (U_1^B, \ldots, U_m^B)$ be group preference models, and $\mathcal{U}_{A \cup B} = (U_1^A, \ldots, U_n^A, U_1^B, \ldots, U_m^B)$. Given a group preference model $\mathcal{U}$, I refer to the SPO corresponding to the aggregation of the preferences in $\mathcal{U}$ as $\succ_{\mathcal{U}}^{\uplus} = \uplus(\succ_{U_1}, \ldots, \succ_{U_n})$.

4.3.1 Semantic Properties of Voting-Based Strategies

Unanimous Winner (UW): If an element a is in a k-rank answer to Q for each $U_i \in \mathcal{U}$, then there exists a k-rank answer to Q for $\mathcal{U}$ that contains element a.

Unanimous Loser (UL): If element a is in none of the k-rank answers to Q for each $U_i \in \mathcal{U}$, then a does not belong to any k-rank answer to Q for $\mathcal{U}$.

In the following, let $\mathcal{U}_{add}$ be a group preference model obtained from $\mathcal{U}$ by adding a new element c to choose from and $\mathcal{U}_{rem}$ be a group preference model obtained from $\mathcal{U}$ by removing from consideration an element c.

Stability 1 (S1): For each k-rank answer $r = \langle a_1, \ldots, a_k \rangle$ to Q for $\mathcal{U}$, there exists a

k-rank answer $r' = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{U}_{add}$ such that either $r = r'$ or (i) $\{b_1, \ldots, b_k\} \setminus \{a_1, \ldots, a_k\} = \{c\}$ (i.e., for some j, $1 \leq j \leq k$, $b_j = c$) and (ii) for each b_i, b_j such that $1 \leq i < j \leq k$, $b_i = a_{i'}$ and $b_j = a_{j'}$, it holds that $i' < j'$.

This property states that whenever a new element is added, the order must be the same among the elements in the result.

Stability 2 (S2): If for each k-rank answer $r = \langle a_1, \ldots, a_k \rangle$ to Q for $\mathcal{U}$, it holds that $a_i \neq c$ for all $1 \leq i \leq k$, then $r' = \langle b_1, \ldots, b_k \rangle$ is a k-rank answer to Q for $\mathcal{U}_{rem}$ if it is a k-rank answer to Q for $\mathcal{U}$.

If an element not in the k-rank is removed from consideration, then the result must be the same.

Monotonicity 1 (M1): Let $r = \langle a_1, \ldots, a_{i-1}, c, a_{i+1}, \ldots, a_k \rangle$ be a k-rank answer to Q for some $U_i \in \mathcal{U}$, and $\mathcal{U}'$ be equal to $\mathcal{U}$ except that one of the U_i's is changed to U_i' such that there exists a k-rank answer $r' = \langle a_1', \ldots, a_{j-1}', c, a_{j+1}', \ldots, a_k' \rangle$ to Q with $j \leq i$. Then, there exists a k-rank answer $r'' = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{U}'$ such that $b_m = c$, for some $1 \leq m \leq k$.

Intuitively, Monotonicity 1 states that if an element is in a k-rank, it is still in a k-rank if a profile changes to rank it higher.

Monotonicity 2 (M2): Let c be an element such that for every k-rank answer $r = \langle a_1, \ldots, a_k \rangle$ to Q for $\mathcal{U}$ we have $c \neq a_i$ for $1 \leq i \leq k$, and let $\mathcal{U}'$ be equal to $\mathcal{U}$ except that one of the U_i's is changed to U_i' such that for every k-rank answer $r' = \langle a_1', \ldots, a_k' \rangle$ to Q for U_i' we have that $c \neq a_i'$ for $1 \leq i \leq k$. Then, every k-rank answer $r = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{U}'$ is such that $c \neq b_i$ for $1 \leq i \leq k$.

If an element is not in the k-rank, then it is not in the k-rank when some profile is changed to lower its rank.

Non-dictatorship (ND): For any U_i in $\mathcal{U}$, there exists $\mathcal{U}' = (U_1', \ldots, U_{i-1}', U_i, U_{i+1}', \ldots, U_n')$ such that there is some k-rank answer to Q for $\mathcal{U}'$ that is not a k-rank answer to Q for $\mathcal{U}$.

	UW	UL	S1	S2	M1	M2	ND
P	✓	✓	×	×	✓	✓	✓
F	×	✓	×	✓	✓	✓	× $(k = 1)$ / ✓$(k \neq 1)$
PM	✓	✓	×	×	✓	✓	✓
FM	×	✓	×	✓	✓	✓	✓

Figure 4.7: Summary of properties satisfied by each voting-based strategy.

Intuitively, there is no U_i in $\mathcal{U}$ such that the k-rank answers to Q for $\mathcal{U}$ are determined by that preference model alone.

Proposition 4.3.1. The following properties hold for the specific voting-based strategies:

- *Plurality* satisfies *UW*, *UL*, *M1*, *M2*, and *ND*.
- *Fairness* satisfies *UL*, *S2*, *M1*, *M2*, and *ND* with $k \neq 1$.
- *Plurality with misery* satisfies *UW*, *UL*, *M1*, *M2*, and *ND*.
- *Fairness with misery* satisfies *UL*, *S2*, *M1*, *M2*, and *ND*.

Proof.

Results for *Unanimous Winner* (UW)

- *Plurality* satisfies UW.

  Proof: If everyone in a group of n users has element a in a k-rank answer, then element a has n votes. The only way that no ranking with element a exists is that there are k elements with at least $n + 1$ votes, which is impossible. □

- *Fairness* does not satisfy UW.

  Proof by counterexample: Suppose we have two group preference models $\mathcal{U}_1 = \langle U_1 \rangle$ and $P_2 = \langle U_2 \rangle$, a query Q, and the following are k-rank answers to Q for each of them with $k = 3$:

$\mathcal{U}_1$:	*Element*
	b
	a
	c

$\mathcal{U}_2$:	*Element*
	d
	e
	c

Element c belongs to a 3-rank answer to Q for $\mathcal{U}_1$ and $\mathcal{U}_2$; however, no matter which order we take between $\mathcal{U}_1$ and $\mathcal{U}_2$, no 3-rank answer to Q for $\mathcal{U}_1 \cup \mathcal{U}_2$ contains element c. □

- *Plurality with misery* satisfies UW.

  Proof: Analogous to the proof for plurality; note that if an element is an unanimous winner, then it cannot be a misery element. □

- *Fairness with misery* does not satisfy UW.

  Proof by counterexample: Consider the same counterexample for the proof that fairness does not satisfy UW. □

Results for *Unanimous Loser* (UL):

- *Plurality* satisfies UL.

  Proof: If element a is in none of the individual k-rank answers, then it has zero votes in the final k-rank answer. By hypothesis, there are at least k elements in the union of all the k-rank answers that have non-zero votes, which means that they will beat element a in any final ranking. □

- *Fairness* satisfies UL.

  Proof: If the element does not appear in any k-rank answer, it is impossible for it to be chosen for the final one. □

- *Plurality with misery* satisfies UL.

Proof: Analogous to plurality — it does not matter if the element is a misery element or not. □

- *Fairness with misery* satisfies UL.

  Proof: If the element does not appear in any k-rank answer, it is impossible for it to be chosen for the final one; note that it does not matter if the element is a misery element or not. □

Results for *Stability 1* (S1):

- *Plurality* does not satisfy *S1*.

  Proof by counterexample: Consider a set of four users and $k = 3$, and let the votes for each user be as follows (such that each user voted elements from their respective skylines):

	$\mathcal{U}_1$:	$\mathcal{U}_2$:	$\mathcal{U}_3$:	$\mathcal{U}_4$:
Element	*Element*	*Element*	*Element*	*Element*
1	b	b	b	b
2	c	c	c	f
3	d	d	h	i

  In this case, the only possible k-rank answer is $\langle b, c, d\rangle$. Now, suppose that element a is added to each user's preferences in such a way that user 1 prefers b, c, and d over a, user 2 prefers a over d, user 3 prefers a to b, and user 4 also prefers a to b. We now have the following votes:

	$\mathcal{U}_1$:	$\mathcal{U}_2$:	$\mathcal{U}_3$:	$\mathcal{U}_4$:
Element	*Element*	*Element*	*Element*	*Element*
1	b	b	a	a
2	c	c	c	f
3	d	a	h	i

  We now have two possible answers: $\langle c, a, b\rangle$ or $\langle a, c, b\rangle$, which violates property $S1$. □

- *Fairness* does not satisfy S1.

  Proof: Consider the same counterexample as for plurality, taking the order P_4, P_3, P_2, P_1. For the answer $\langle b, c, d\rangle$, every possible answer after adding a either does not include b, or it includes it in the last place; therefore, property $S1$ is not satisfied. □

- *Plurality with misery* does not satisfy S1.

  Proof: Consider the same counterexample as for plurality, and a is not a misery element. □

- *Fairness with misery* does not satisfy S1.

  Proof: Consider the same counterexample as for fairness, and a is not a misery element. □

Stability 2 (S2):

- *Plurality* does not satisfy S2.

  Proof by counterexample: Consider a set of six users and $k = 2$, and let the votes for each user be as follows:

	Element
$\mathcal{U}_1$:	b
	a

	Element
$\mathcal{U}_2$:	b
	c

	Element
$\mathcal{U}_3$:	d
	c

	Element
$\mathcal{U}_4$:	d
	c

	Element
$\mathcal{U}_5$:	b
	a

	Element
$\mathcal{U}_6$:	b
	f

  In this case, the only possible k-rank is $\langle b, c\rangle$. Now, suppose that element a is removed, and the new votes are as follows:

	Element		*Element*		*Element*		*Element*	
$\mathcal{U}_1$:	b	$\mathcal{U}_2$:	b	$\mathcal{U}_3$:	d	$\mathcal{U}_4$:	d	
	d		c		c		c	
	Element		*Element*		*Element*			
$\mathcal{U}_5$:	b	$\mathcal{U}_6$:	b					
	d		f					

Now, the only possible k-rank answers are $\langle b, d\rangle$ or $\langle d, b\rangle$, which violates S2. □

- *Fairness* satisfies S2.

  Proof: Assume towards a contradiction that some answer to the query for $\mathcal{U}_{rem}$ is not an answer for $\mathcal{U}$. This means that the removal of element c made another element e (that does not appear in any answer for $\mathcal{U}$) available for voting in the fairness round, for at least one user. But this would imply that c itself was available for $\mathcal{U}$ at the point, in which e becomes available for $\mathcal{U}_{rem}$; this contradicts the assumption that c does not appear in any answer for $\mathcal{U}$. □

- *Plurality with misery* does not satisfy S2.

  Proof: Consider the same example as for plurality, assuming that element a is not a misery element. □

- *Fairness with misery* satisfies S2.

  Proof: If the element is a misery element, the result will be the same. Otherwise, it is analogous to fairness. □

Results for *Monotonicity 1 (M1)*:

- *Plurality* satisfies M1.

  Proof: Trivial (the element can only gain votes). □

- *Fairness* satisfies M1.

  Proof: If the element was chosen, improving its position in some rankings can only help it to be chosen earlier. □

- *Plurality with misery* satisfies M1.

  Proof: Analogous to the proof for plurality, since if the element is chosen, it cannot be a misery element. □

- *Fairness with misery* satisfies M1.

  Proof: Same argument as for fairness; note that if the element is chosen, it cannot be a misery element. □

Monotonicity 2 (M2):

- *Plurality* satisfies M2.

  Proof: Trivial (the element can only lose votes). □

- *Fairness* satisfies M2.

  Proof: If an element is not chosen, ranking it even lower cannot cause it to be chosen. □

- *Plurality with misery* satisfies M2.

  Proof: If the element is a misery element, the result will be the same. Otherwise, the proof is analogous to that for plurality. □

- *Fairness with misery* satisfies M2.

  Proof: Same argument as for fairness; note that if the element is a misery element, the result will be the same. □

Non-dictatorship (ND):

- *Plurality* satisfies *ND*.

  Proof: One user can only influence the choice by adding k votes (one to each element in their ranking). Therefore, a single user cannot dictate the outcome of the result. □

- *Fairness* satisfies ND for any $k \neq 1$.

  Proof: If $k = 1$, the first user to choose is the dictator. Otherwise, the result is determined by at least two users. □

- *Plurality with misery* satisfies ND.

  Proof: Same argument as that of plurality. □

- *Fairness with misery* satisfies ND.

  Proof: Though the scenario for $k = 1$ could cause the first user to choose to be a dictator, that user's choice might be in another user's misery list. □

4.3.2 Semantic Properties of the CSU Strategy

Group Union Preservation First Position (GUPF): Let $r = \langle a_1, \ldots, a_k \rangle$ and $r' = \langle b_1, \ldots, b_k \rangle$ be k-rank answers to Q for $\mathcal{U}_A$ and $\mathcal{U}_B$, respectively. If $a = a_1 = b_1$, then there exists a k-rank answer $r'' = \langle a, c_1, \ldots, c_{k-1} \rangle$ to Q for $\mathcal{U}_{A \cup B}$.

Intuitively, if element a appears in the first position of a k-rank answer to Q for both $\mathcal{U}_A$ and $\mathcal{U}_B$, then a appears in the first position of a k-rank answer to Q for the union of the group preference models.

UW First Position (UWF): If for every U_i in $\mathcal{U}$, there exists a k-rank answer to Q for U_i of the form $\langle a, b_1, \ldots, b_{k-1} \rangle$, then there exists a k-rank answer to Q for $\mathcal{U}$ of the form $\langle a, c_1, \ldots, c_{k-1} \rangle$.

Intuitively, if element a appears in the first position of a k-rank answer to Q for each $U_i \in \mathcal{U}$, then a appears in the first position of a k-rank answer to Q for $\mathcal{U}$.

Unanimity on preference relation (UP): If removeCycles only removes edges (v_1, v_2) whenever there does not exist another edge in the cycle labelled with a lower number, then whenever $t_1 \succ_{U_i} t_2$ for each U_i in $\mathcal{U}$, then $t_1 \succ_{\mathcal{U}}^{\uplus} t_2$.

Proposition 4.3.2. The CSU strategy (as implemented in Algorithm AggPrefsCSU) satisfies *UP*. Furthermore, if the graph G in AggPrefsCSU does not contain cycles after exiting the for-loop in line 3, then it also satisfies *GUPF* and *UWF*.

- *CSU* satisfies UP.

  Proof: Follows directly from condition (ii) in Theorem 4.2.1. Note that if there is a cycle, there must exist an edge in the cycle with a lower number of votes than the removed edge. □

- *CSU* satisfies GUPF.

  Proof: If element a is in the first position of a k-rank answer to both $\mathcal{U}_A$ and $\mathcal{U}_B$, then it belongs to the skylines of both $\mathcal{U}_A$ and $\mathcal{U}_B$; given that there are no cycles in the graph, this means that it also belongs to the skyline of $\mathcal{U}_{A \cup B}$. Therefore, there exists a k-rank answer to Q for $\mathcal{U}$ that contains a in the first position. □

- *CSU* satisfies UWF.

  Proof:

  Direct consequence of the fact that *CSU* satisfies GUPF. □

4.4 Implementation and Evaluation

This section evaluates and analyses the running time of the proposed algorithms and the quality of their results, over experiments done with real-world data.

The feasibility of the approach is demonstrated by applying it to preference models obtained from real users.

When building a new formalism, one of the problems that arises is the lack of datasets to test empirically your formalism. Since preferences are private information, this data is very difficult to find. Therefore, I gathered the preferences of 50 users, using a Web application. Users stated their preferences as SPOs, over cuisine, type of food, and type of place (breakfast, lunch, and dinner). Users entered their preferences (e.g., I prefer Italian food over Japanese food). Based on the fact that users know each other, I created 19 groups of 3 to 9 users.

The code of the algorithms and dataset are available as open source [156].

All runs were carried out using the Datalog$^{\pm}$ ontology built from the Yelp Dataset Challenge [175], which contains 11,537 businesses in the Phoenix (USA) metropolitan area. The queries represent situations where groups wish to decide where to go for a meal.

I discuss three research questions: "Q1: *Which approach is more efficient?*" ,"Q2: *Which aggregation method yields the best results*?"', and "Q3: *How different are the results produced by each aggregation method?*".

4.4.1 Implementation and Hardware

I have implemented GPP_Datalog$^{\pm}$ by extending the Datalog$^{\pm}$ query answering engine in [77], which supports FO-rewritable fragments of the Datalog$^{\pm}$ family of ontology languages. Essentially, the extension involved adding query answering based on group preferences, as well as handling of merging and aggregation operators. As for the latter, I have implemented collapse to single user (CSU), plurality voting, and plurality voting with misery. All graph operations were implemented using the JGraphT library (`http://jgrapht.org/`), which provides efficient data structures for the representation of graph structures as well as efficient implementations of operations such as reachability and cycle detection. The entire implementation was done in Java.

All runs were done on an Intel Core i7 processor at 2.2 GHz and 8GB of RAM,

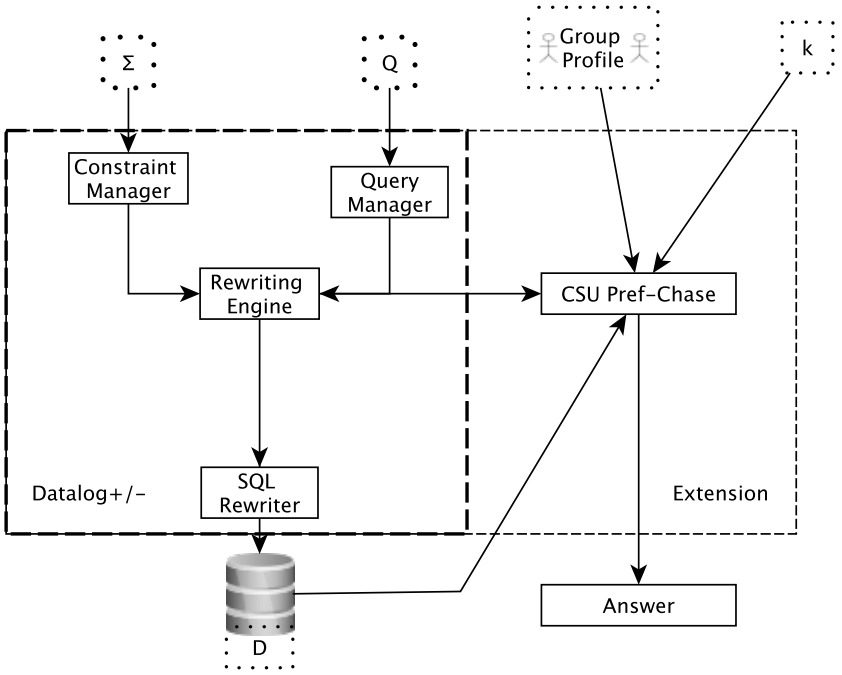


Figure 4.8: The architecture of the system.

under the Mac OS X 10.6.8 operating system, and a Sun JVM Standard Edition with maximum heap size set at 512 MB of RAM. To minimise experimental variation, all results are averages of three independent runs.

4.4.2 Experimental Setup

Inputs to the system consist of tuples of the form $\langle Q, O, \mathcal{P}, k\rangle$ with $O = (D, \Sigma)$, where Q is a query, D is a database, Σ is a finite set of TGDs, $\mathcal{P}$ is a group preference model, and k is the number of query results. The CSU-PrefChase consists of a graph that stores all user preferences after constructing the chase. The preference graph is a labelled directed multi-graph (N, E, ℓ), where N is the node set (the atoms), E is the edge set (the preference relation), and the labelling function ℓ stores for each edge the identity of its user. That is, edges in E are pairs of nodes (u, v), which state that u is preferred to v by the user specified in the label $\ell(u, v)$. CSU-PrefChase is adapted for obtaining the answer to all queries, depending on the chosen strategy.

A high-level overview of the architecture of the system can be found in Figure 4.8.

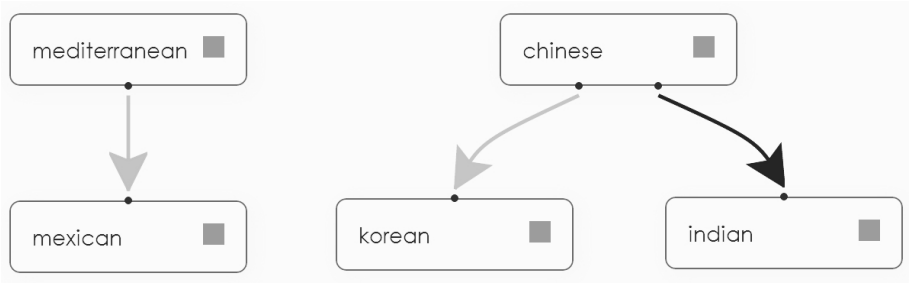


Figure 4.9: Preferences of a user for dinner (e.g., prefers Mediterranean food over Mexican food).

$$
\begin{aligned}
\Sigma = \{\ & barbecue(X) \rightarrow food(X), && local\_flavor(X) \rightarrow food(X),\\
& hot\_dogs(X) \rightarrow food(X), && vegetarian(X) \rightarrow food(X),\\
& seafood(X) \rightarrow food(X), && chicken\_wings(X) \rightarrow food(X),\\
& pizza(X) \rightarrow food(X), && dessert(X) \rightarrow food(X),\\
& sandwiches(X) \rightarrow food(X), && gluten\_free(X) \rightarrow food(X),\\
& burgers(X) \rightarrow food(X), && fast\_food(X) \rightarrow food(X),\\
& donuts(X) \rightarrow dessert(X), && bagels(X) \rightarrow dessert(X),\\
& ice\_cream(X) \rightarrow dessert(X), && dance\_clubs(X) \rightarrow place(X),\\
& juice\_bars(X) \rightarrow place(X), && cafes(X) \rightarrow place(X),\\
& bakeries(X) \rightarrow place(X), && breweries(X) \rightarrow place(X),\\
& steakhouses(X) \rightarrow place(X), && lounges(X) \rightarrow place(X),\\
& buffets(X) \rightarrow place(X), && bars(X) \rightarrow place(X),\\
& dive\_bars(X) \rightarrow bars(X), && pubs(X) \rightarrow bars(X),\\
& sports\_bars(X) \rightarrow bars(X), && sushi\_bars(X) \rightarrow bars(X),\\
& wine\_bars(X) \rightarrow bars(X), && mexican(X) \rightarrow cuisine(X),\\
& american(X) \rightarrow cuisine(X), && local\_services(X) \rightarrow cuisine(X),\\
& asian(X) \rightarrow cuisine(X), && middle\_eastern(X) \rightarrow cuisine(X),\\
& mediterranean(X) \rightarrow cuisine(X), && chinese(X) \rightarrow asian(X),\\
& indian(X) \rightarrow asian(X), && thai(X) \rightarrow asian(X),\\
& korean(X) \rightarrow asian(X), && japanese(X) \rightarrow asian(X),\\
& vietnamese(X) \rightarrow asian(X), && greek(X) \rightarrow mediterranean(X),\\
& italian(X) \rightarrow mediterranean(X), && french(X) \rightarrow mediterranean(X),\\
& food(X) \rightarrow \exists Y\, hasName(X,Y), && food(X) \rightarrow \exists Y\, inCity(X,Y),\\
& food(X) \rightarrow \exists Y\, inState(X,Y), && cuisine(X) \rightarrow \exists Y\, hasName(X,Y),\\
& cuisine(X) \rightarrow \exists Y\, inCity(X,Y), && cuisine(X) \rightarrow \exists Y\, inState(X,Y),\\
& place(X) \rightarrow \exists Y\, hasName(X,Y), && place(X) \rightarrow \exists Y\, inCity(X,Y),\\
& place(X) \rightarrow \exists Y\, inState(X,Y)\} &&
\end{aligned}
$$

Figure 4.10: Set of TGDs for the ontology used in the experimental evaluation.

The input consists of a tuple of the form $\langle Q, O, \mathcal{P}, k\rangle$ with $O = (D, \Sigma)$, where Q is a query, D is a database, Σ is a finite set of TGDs, $\mathcal{P}$ is a group preference model, and k is the number of query results. The set of TGDs Σ for the ontology in the experiments is given in Figure 4.10, while the structure of the database for this ontology is shown in Figure 4.18. Note that this ontology is different from the one that I used in the examples throughout the chapter. The constraint manager, the query manager, and the rewriting engine are directly taken from the Datalog$^{\pm}$ system of [77]: the constraint manager checks unions of conjunctive queries that are used to check if D satisfies Σ; the query manager schedules queries for rewriting and execution; finally, the rewriting engine converts the input query into a union of conjunctive queries after all the TGDs are applied. The list of queries obtained after the rewriting step is the input for the CSU-PrefChase subsystem, which constructs the preferences over the atoms that are answers to this query.

The CSU-PrefChase builds a graph that stores all the preferences of the users after the standard chase is computed. The preference graph is a labelled directed graph (N, E, ℓ), where N is the node set (the atoms), E is the edge set (the preference relation), and the labelling function ℓ stores for each edge the identity of its user. That is, edges in E are pairs of nodes (u, v), which state that u is preferred over v by the user specified in the label $\ell(u, v)$. Note that CSU-PrefChase is used not only for the CSU strategy, but also for the plurality with misery and fairness with misery strategies — the latter strategies involve calling CSU-PrefChase individually for each user to obtain the result of merging the users' SPOs with the one derived from the probabilistic model.

Data. All runs were carried out using an ontology built on the basis of the data from the Yelp Dataset Challenge [175]; this dataset contains $11,537$ businesses in the Phoenix metropolitan area in the United States, $8,282$ check-in sets, $43,873$ users, and $229,907$ reviews — each business has one or more associated categories. The Datalog$^{\pm}$ ontology used for these experiments was constructed as follows. I

City	# of Businesses	Average # of Nodes	Average # of Pref. Edges
Peoria	109	42.33	1,136.53
Gilbert	163	61.68	2,872.91
Glendale	242	91.93	7,171.96
Chandler	349	136.17	16,222.71
Tempe	465	180.97	27,835.80

Figure 4.11: Size information for the different subsets of the used dataset and corresponding average size of the CSU-PrefChase built during the query answering process (using the CSU aggregation strategy).

manually created the TGDs using categories related to the dataset, yielding a set of 53 TGDs. For example, the "Asian" and "Chinese" categories give rise to the TGD $\mathit{chinese}(X) \rightarrow \mathit{asian}(X)$. The database instance was automatically generated using the mapping between businesses and categories. To test the algorithms on instances of different sizes, I considered five different subsets of the dataset corresponding to different cities in the state of Arizona (cf. Figure 4.11). Finally, the database for this ontology was stored in a PostgreSQL 9.3 database. Figure 4.10 presents the underlying ontology.

User Preferences. I requested users to define preferences over places along two dimensions: meal (breakfast, lunch, and dinner), details of the meal: *cuisine* (Italian, Asian, etc.), *type of food* (pizza, sandwich, etc.), and *type of place* (bar, cafe, etc.) — each combination of these produced a set of nine so-called *choice scenarios*, such as "cuisine for lunch" or "type of food for dinner". Preferences were entered using a graphical interface illustrated in Figure 4.9. A total of 49 people responded to the request, yielding a total of 364 SPOs (not all users entered their preferences for all nine scenarios). These SPOs were then processed using the set of businesses in the dataset in such a way that business X is preferred over business Y iff the SPO establishes the preference over their corresponding categories (for instance, if the users specified that they prefer bagels over sushi, then all bagel businesses are preferred over all sushi businesses).

Group Definition. The groups of users were created manually, such that all members know each other. From the total of 49 users, I generated 19 different groups, ranging from 3 to 9 users. This produced a grand total of $19 \cdot 9 = 171$ group choice scenarios; for each data point in the group size variation experiments below, ten SPOs have been randomly chosen and the average has been reported.

Probabilistic Model and Choice of Parameters. For each business, the Yelp dataset provides a numerical rating ranging from 0 to 5. I used this information to create a probabilistic model by using the category supplied in the dataset; the probability assigned to a business in a category is computed as $(\mathit{rating} + 0.5)/5.5$ if the categories match, and zero otherwise. Though more complex models are of course possible, this is outside the scope of the experiments. Finally, values for the parameter t were generated automatically as uniformly distributed random values in the interval $[m - \sigma, m + \sigma]$, where m is the mean value of the ratings, and σ is the standard deviation.

Queries. I used the following three queries: $Q_1(X) = \mathit{food}(X)$, $Q_2(X) = \mathit{cuisine}(X)$, and $Q_3(X) = \mathit{place}(X)$, representing the situations where groups wish to decide about where to go to eat, based on the preferences over kinds of food, cuisine, and place, respectively. Unless stated otherwise, all runs involve k-rank queries with $5 \leq k \leq 20$, in steps of 5.

4.4.3 Performance Evaluation

The main performance metric that I investigate in this study is the time that is required to carry out query answering when certain parameters are varied. **??**, left side, shows scatterplots for the running time when three different parameters are varied: aggregation strategy (one graph per strategy), group size (x axis), and number of businesses in the database (different markers in each graph); as described above, the numbers for the latter arise from the selection of five different cities that were chosen in order to evaluate how running times vary when the number of businesses

increases gradually. Note that groups of size 6 do not have data points associated with them — this is due to the fact that the data did not contain enough instances of combinations of groups of this size with choice scenarios. The dotted/dashed lines in these graphs correspond to trend lines showing the increase in running time when the group size increases.

The graphs in Figure **??**, right side, show how the running time increases as the number of businesses in the database increases (each point is an average over the group sizes considered) — note that the y axis in these figures is on a logarithmic scale. These results show that the prototype implementation yields a top-3 answer for a city with 465 businesses to choose from in about 27 seconds. The table in Figure 4.11 shows the number of nodes in the preference chase for each city, as well as the number of edges for the collapsed graph when the CSU strategy is used.

The results in Figure **??** show that the three different strategies have similar running times for these runs; however, the way in which the running time is used by the two main approaches (CSU and plurality) is different. To illustrate this difference, Figure 4.14 plots the partial times in pie charts; for instance, observe that CSU takes more time computing the top-k, while plurality takes more time computing the result of the merging operator.

4.4.4 Quality Evaluation

In this section, I discuss experiments carried out to evaluate the quality of the results produced by my algorithms. I first discuss the metrics used, and then go on to discuss the results. The experimental setup is identical to the one used in the previous experiments, except that I focused on a single city (*Gilbert*), since the number of businesses is not varied.

4.4.4.1 Evaluation Metrics

Usually, methods based on precision and recall are used when comparing two lists of results — however, since these methods rely on the availability of "ground truth"

and there is no clear notion of what such a ground truth is for my setting, these methods are not applicable. Therefore, I use quality measurements that are often applied in evaluating information retrieval and group recommender systems [62, 135]; the main ones adopted are *tuple agreement*, *Kendall tau distance*, and *Spearman's footrule*.

Given two top-k lists, *tuple agreement* is defined as the number of tuples that appear in both lists (i.e., it does not consider the actual ordering of the tuples). To allow comparisons across different values of k, I normalise the tuple agreement and call this value *Agreement*. Note that *higher is better for Agreement*, a value of 1 indicating lists with the same elements (perhaps in different order). The second measure is a variation of the *Kendall tau distance* for partial orders that computes the distance between two partial rankings based on the number of pairwise disagreements between them [62] — a disagreement receives a penalty of 1, while for agreement there is no penalty. In case the two top-k lists are permutations of each other, this is the number of exchanges required to convert one into the other. However, if a pair (i, j) appears in one list but not in the other, I do not know if the penalty should be 0 or 1; therefore, it takes a parameter p indicating how pessimistic the metric should be ($p = 1$ is most pessimistic). The measure is normalised by the square of the length of the union of the two lists, and this value is called $Kendall_p$. Note that *lower is better for Kendall*, since it measures differences instead of agreement. Finally, the third measure is a variation of the *Spearman's footrule* metric for partial orders that computes the distance between two partial rankings using the difference of positions of elements of one list in comparison with the other [62]. Where *Kendall* just counts number of swaps, this metric counts how far each element must be moved to reach the place occupied in the other list. This measure is normalised as before, and the result is called *Spearman*. Note that, like *Kendall*, *lower is better for Spearman*.

4.4.4.2 Results

I now discuss two research questions: "*Which aggregation method yields the best results?*" and "*How different are the results produced by each aggregation method?*". I discuss each of these questions in turn.

Question 1: "Which aggregation method yields the best results?" To answer this question, for each group, I computed a top-k query answer r_g, as well as a top-k query answer r_i for each user in the group. I then computed the value of each measure over r_g and r_i, and finally report the mean of all such results. I carried out three different comparisons: overall, varying group size, and varying the value of k — for the first, I report five different measures, while for the other two, we focus on *Agreement* and *Spearman*.

Overall Comparison. Figure 4.15 (left) shows an overall comparison between each aggregation method and the users' rankings. Here, we see that plurality voting performed best in my experiments both in terms of the agreement between the desires of each individual user and what the aggregated result yields (*Agreement* measure), as well as in terms of the ordering of the results (the rest of the measures). Plurality with misery was the worst performer, both in terms of agreement and ordering of the top-k. This is likely due to the fact that this method empowers users to veto options that could be ranked high by several others. Another problem could arise due to cycles; for efficiency, my implementation of *removeCycles* iteratively chooses a random edge from a cycle and removes it without taking into consideration the effect of this operation on the opinions of the other members. Developing a better way to address cycles is an interesting topic of future research.

Varying Group Size. Figure 4.16 shows the results of experiments varying the group size between 2 and 7 (*Agreement* measure on the left, *Spearman* on the right) — I eliminated group sizes, for which I did not have enough data. Results show that all aggregation methods tend to worsen with both measures as group size increases —

this is consistent with the fact that groups of larger size are more difficult to satisfy when k is fixed since it is more likely that conflicting preferences exist.

Varying Value of k. Figure 4.17 shows my results when varying k between 5 and 20 — as before, I have results for the *Agreement* measure on the left and for *Spearman* on the right. Opposite to what was observed when varying group size, we see that increasing the value of k affords an increase of performance for all aggregation methods over both measures. One possible exception is misery with respect to the *Agreement* measure, which seems to slightly worsen as k increases – this special behaviour is likely a consequence of the same issues discussed above. This overall tendency to perform better as k increases makes sense for a group of fixed size, since more space becomes available in the result to accommodate potentially conflicting user preferences.

Question 2: "How different are the results produced by each aggregation method?" Figure 4.15 (right) shows an overall comparison between the results produced by each of the aggregation methods. Focusing on plurality (the overall best performer in the experiments for Question 1), we now compare how similar CSU and misery are to plurality. We can see that with respect to the *Agreement* measure, plurality and CSU are the most similar for these runs; interestingly, for the rest of the measures, the most similar to plurality was misery. Figures 4.19 and 4.20 complement these results — the former shows the same behaviour when varying k for *Agreement*, though for *Spearman*, CSU turns out to be a little better than misery. The latter experiment, however, returns to the same pattern with respect to variations in group size, and CSU is most similar to plurality relative to *Agreement*, while misery is most similar relative to *Spearman* (though the differences are not large at the highest setting of the parameter).

These results suggest that CSU is good at getting the right elements in the result, but misery is good at getting the order right (using the results obtained by plurality as a baseline).

Finally, I ran two-tailed two-sample Student's t-tests comparing all pairs of experiments reported in Figure 4.15. All tests yielded p-values well below 0.001, which shows statistical significance, except for three cases: (i) "CSU vs. User" / "Misery vs. User" for Kendall-0.5 ($p = 0.11008$), (ii) "CSU vs. Misery" / "Plurality vs. Misery" for Agreement ($p = 0.53601$), and (iii) "CSU vs. Plurality" / "CSU vs. Misery" for Kendall-1 ($p = 0.54377$) — note that these higher p-values are expected, given the similarity of the results reported in those cases.

4.5 Conclusion

In this chapter, I have proposed a two-fold extension of the Datalog$^{\pm}$ ontology language: allowing for partially ordered preferences of groups of users, and the management of probabilistic uncertainty. To my knowledge, this is the first combination of ontology languages with group preferences. I focused on answering k-rank queries in this context, and studied different approaches to computing answers to queries based on group preferences; the main challenge is to find a principled way in which to combine the preferences of each individual in the group, while also taking into account the preferences induced by the probabilities obtained from the uncertainty model. I then studied algorithms to answer k-rank queries for disjunctions of atomic queries under these conditions, and showed that it can be done in polynomial time in the data complexity, as long as query answering can also be done so in the underlying classical ontology. Finally, I presented a prototype implementation of my framework, including an empirical analysis using real-world data, and preference models obtained from real users, showing that my approach is feasible in practice.

An interesting question that comes up when implementing these techniques in a real-world scenario is how to decide which aggregation technique to use — this will indeed have an impact on how group decisions are made, as seen in Section 3.4. Some guidelines regarding this question for the techniques studied here are the following: (i) CSU is a fine-grained approach that, given its consideration of all pairs, allows

for a better leveraging of user preference information; (ii) on the other hand, the voting-based approach taken in this chapter computes k-rankings as a first step, which may end up containing arbitrary elements. For instance, coming back to the friends in Example 1.2.1, consider the case in which $k = 2$ and Alberto's preferences yields ranking $\langle r_1, r_3 \rangle$, Bruno's yields $\langle r_2, r_3 \rangle$, and Charles's $\langle r_1, r_2 \rangle$ — note that Charles is the only one who actually cares about the position of the second element, while the answers of the other two were completed arbitrarily after the first position. The *misery* and *fairness* techniques can help mitigate these issues, but in general they can still arise. On the other hand, CSU has the weakness of only working on partial graphs — in carrying out my experiments with real users, I found that people do not find partial orderings of elements natural (most questions were variations of "Do I have to fill in all possible pairs?"). Thus, the voting-based approaches have the advantage of simplicity, since it is possible to directly request a linear ordering of elements from each user, since this is the first step in the aggregation anyway.

Some of the suggestions discussed below on how to continue this research address this issue directly. Also, from my quality evaluation it seems that plurality is the most similar to what individual users want; still, many times users are willing to make compromises depending on other group members and my current approach does not contemplate this.

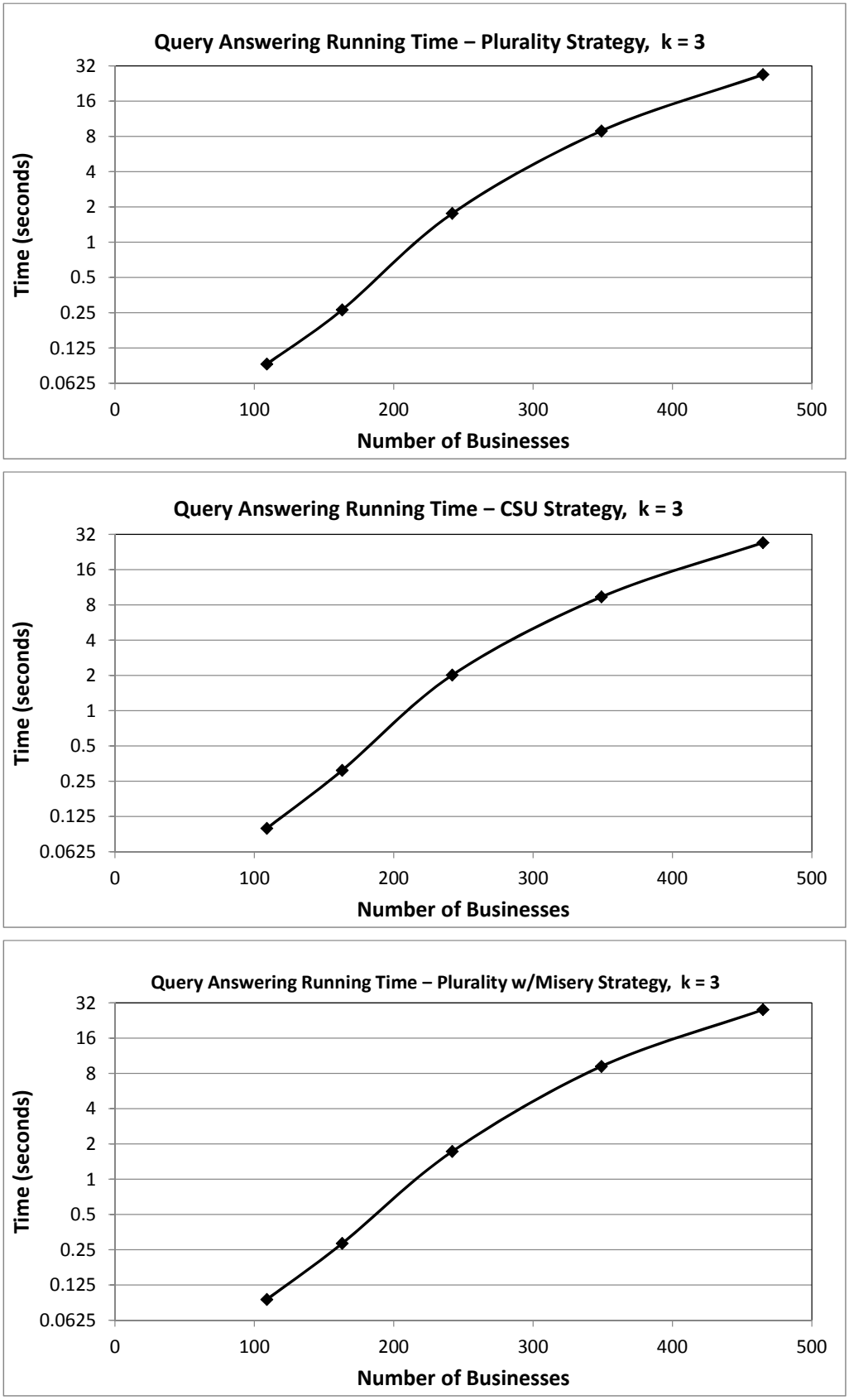


Figure 4.12: Running times for top-3 query answering when varying the number of businesses in the database, the group size, and the preference aggregation strategies.

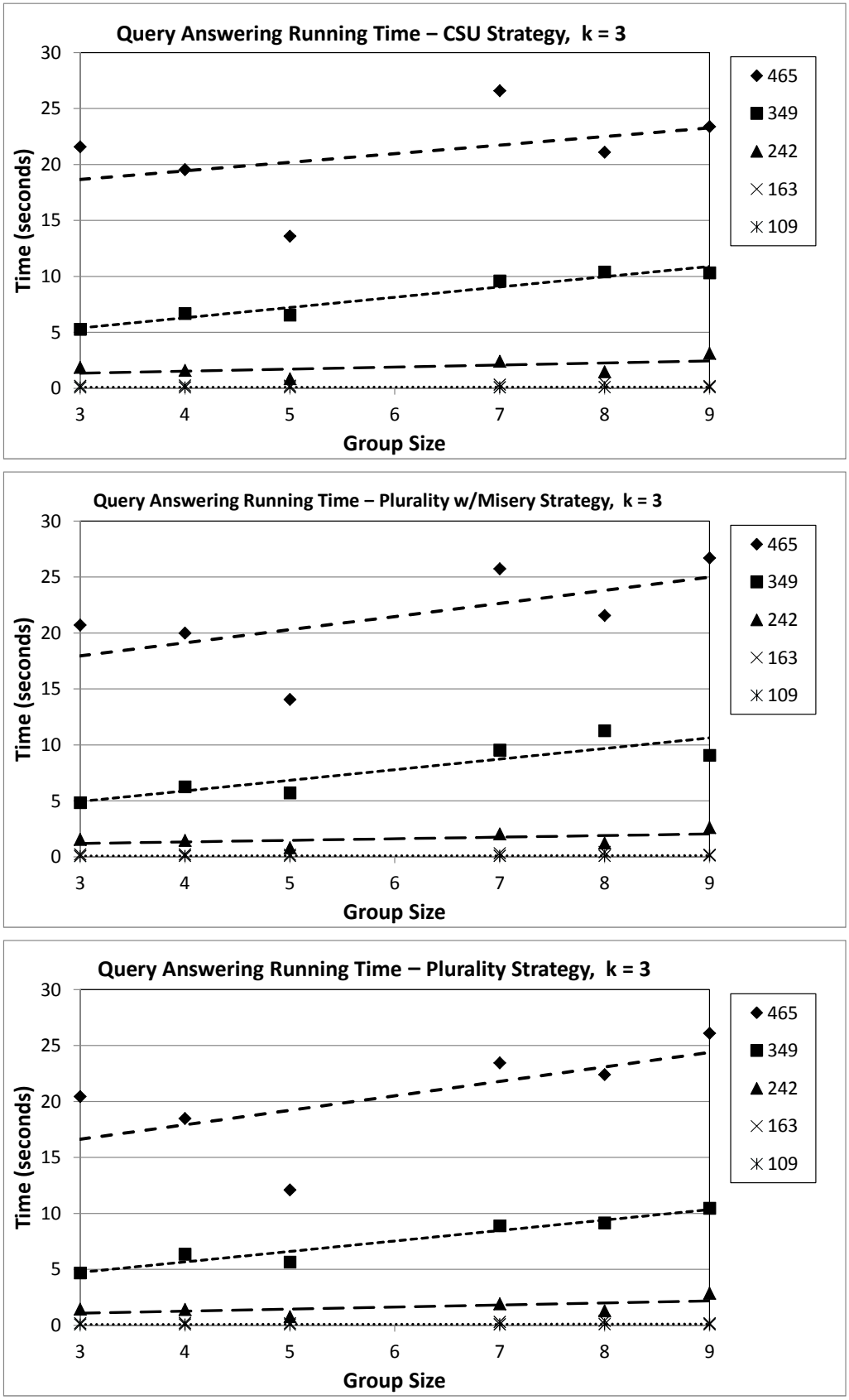


Figure 4.13: Running times for top-3 query answering when varying the number of businesses in the database, the group size, and the preference aggregation strategies.

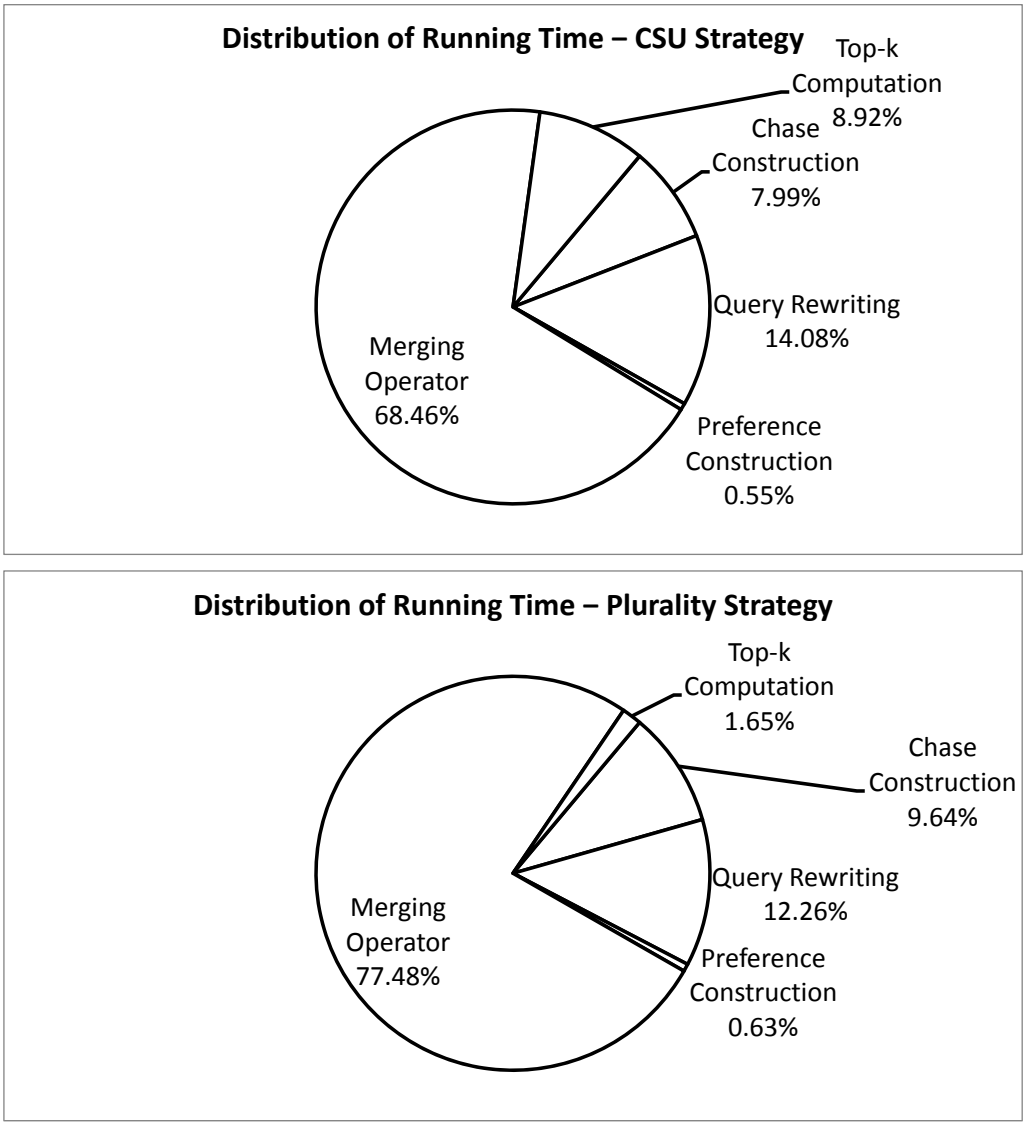


Figure 4.14: Distribution of running times for the CSU and plurality strategies.

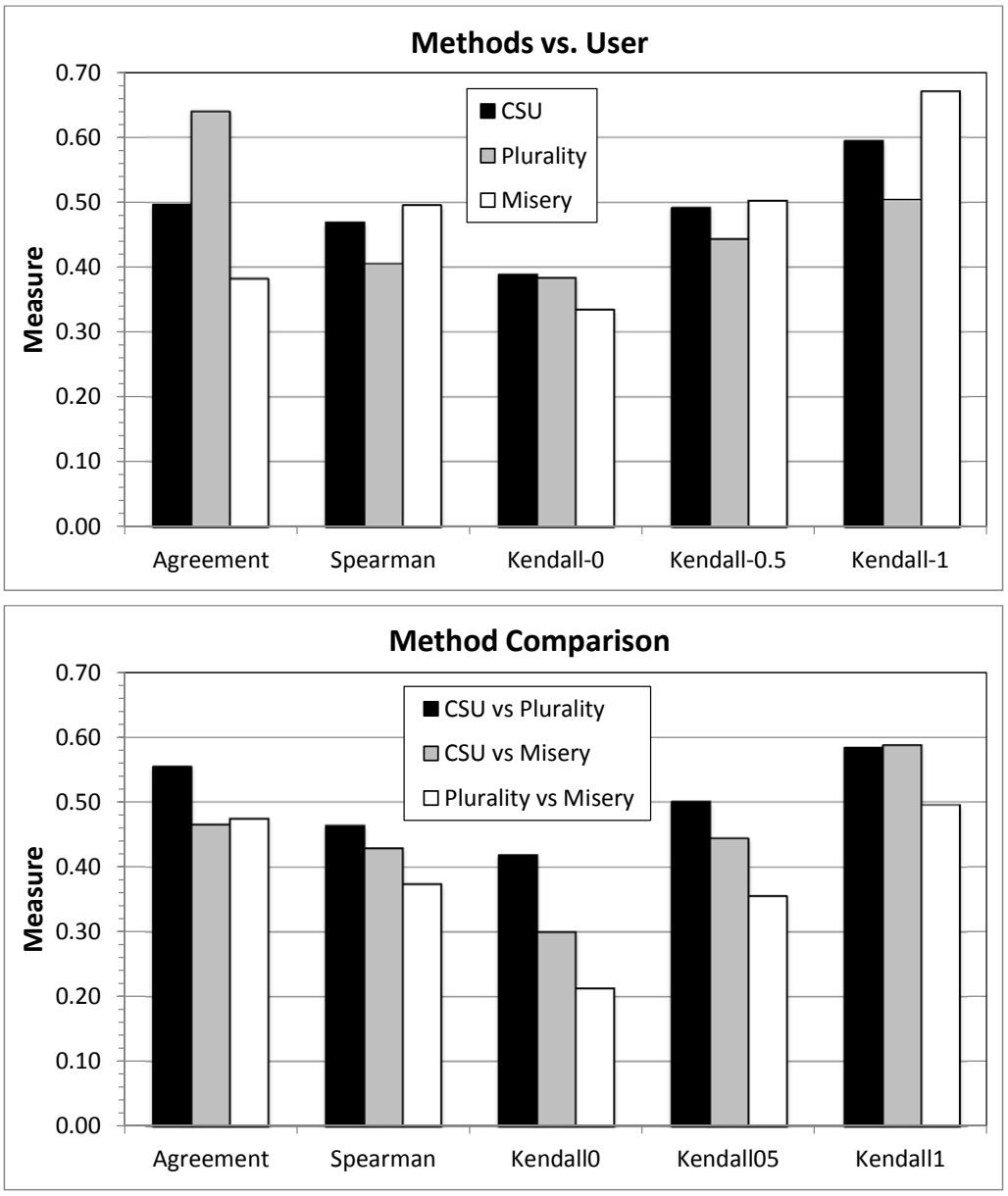


Figure 4.15: Comparison of the approaches with user preferences (left) and pairwise comparison of approaches (right). Higher is better for *Agreement*, while lower is better for the rest.

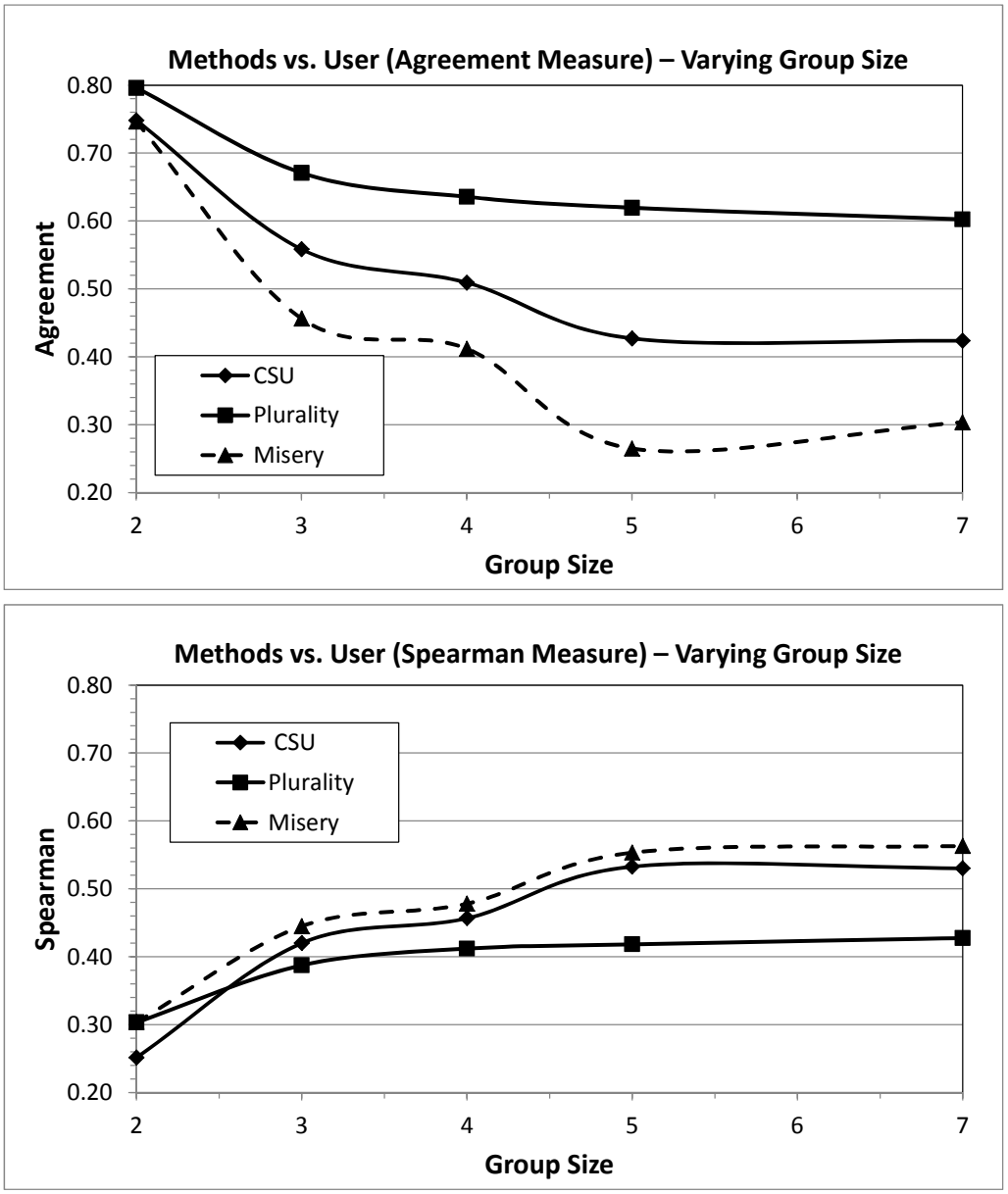


Figure 4.16: How Agreement (left; higher is better) and Spearman (right; lower is better) change depending on group size.

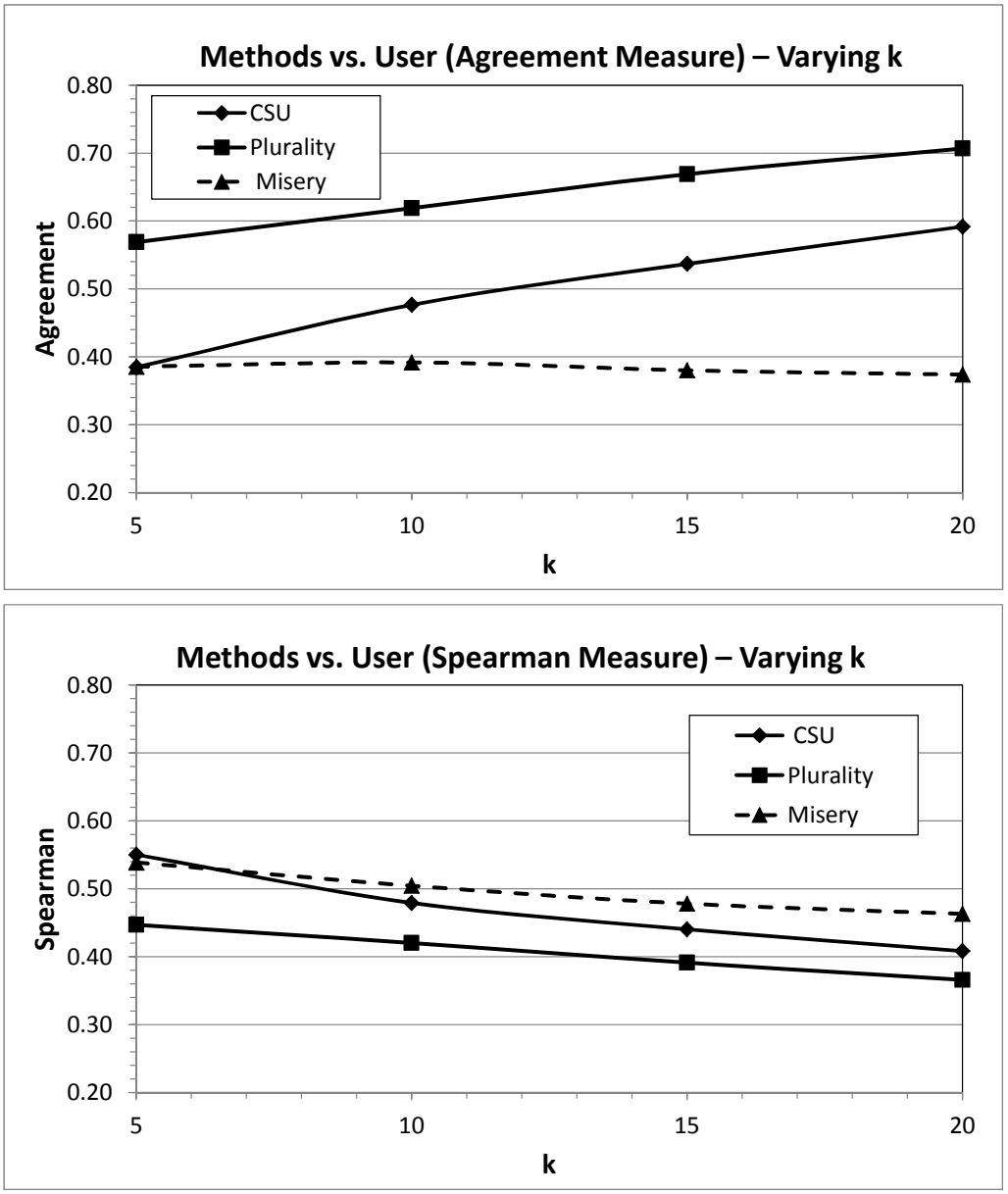


Figure 4.17: How Agreement (left; higher is better) and Spearman (right; lower is better) change depending on the value of k.

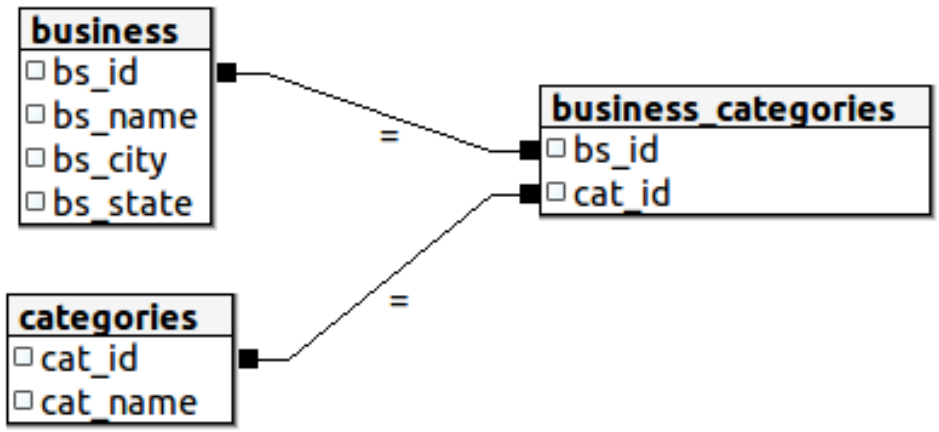


Figure 4.18: The database structure of the Datalog$^{\pm}$ ontology.

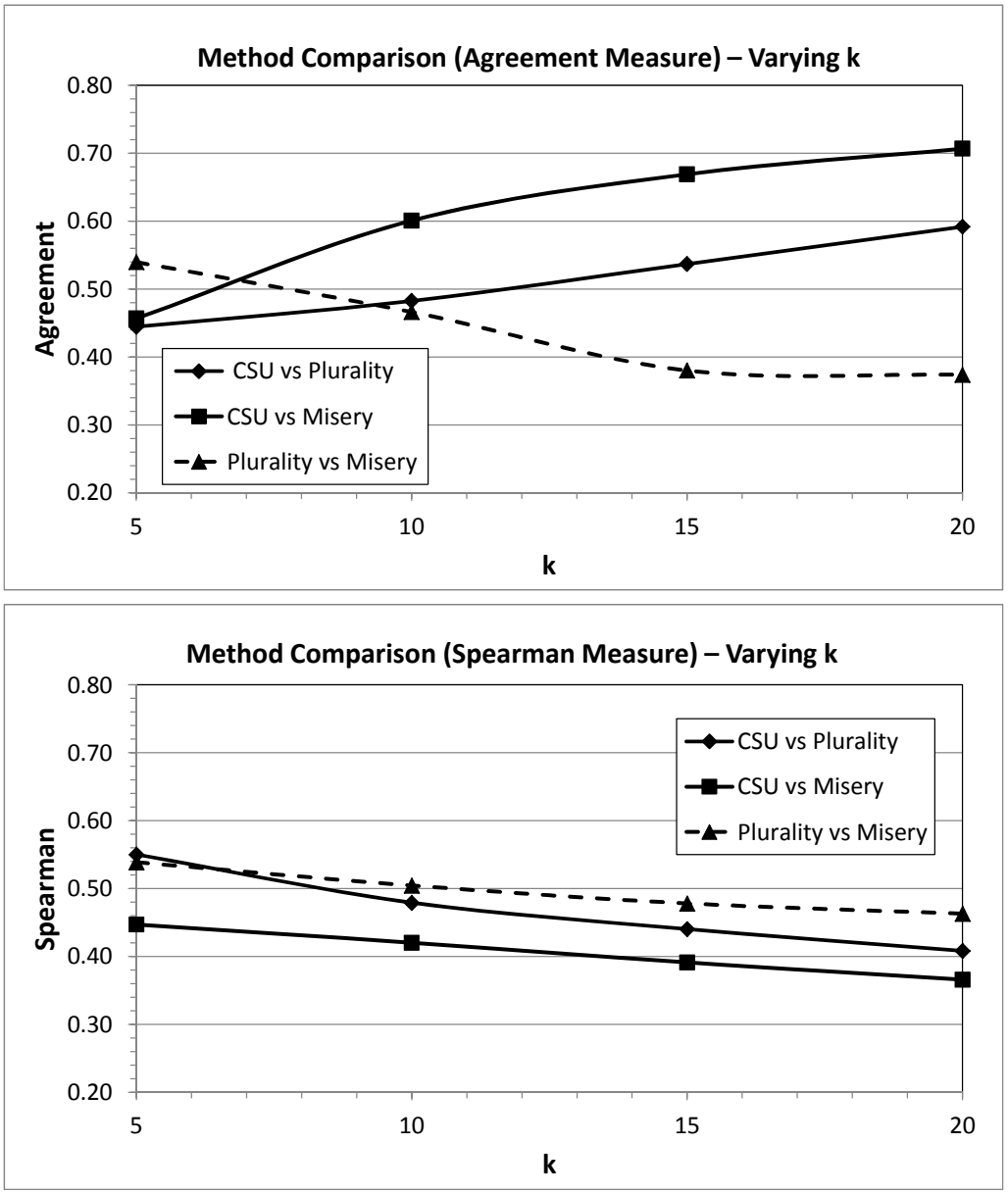


Figure 4.19: How Agreement (left; higher is better) and Spearman (right; lower is better) change depending on the value of k.

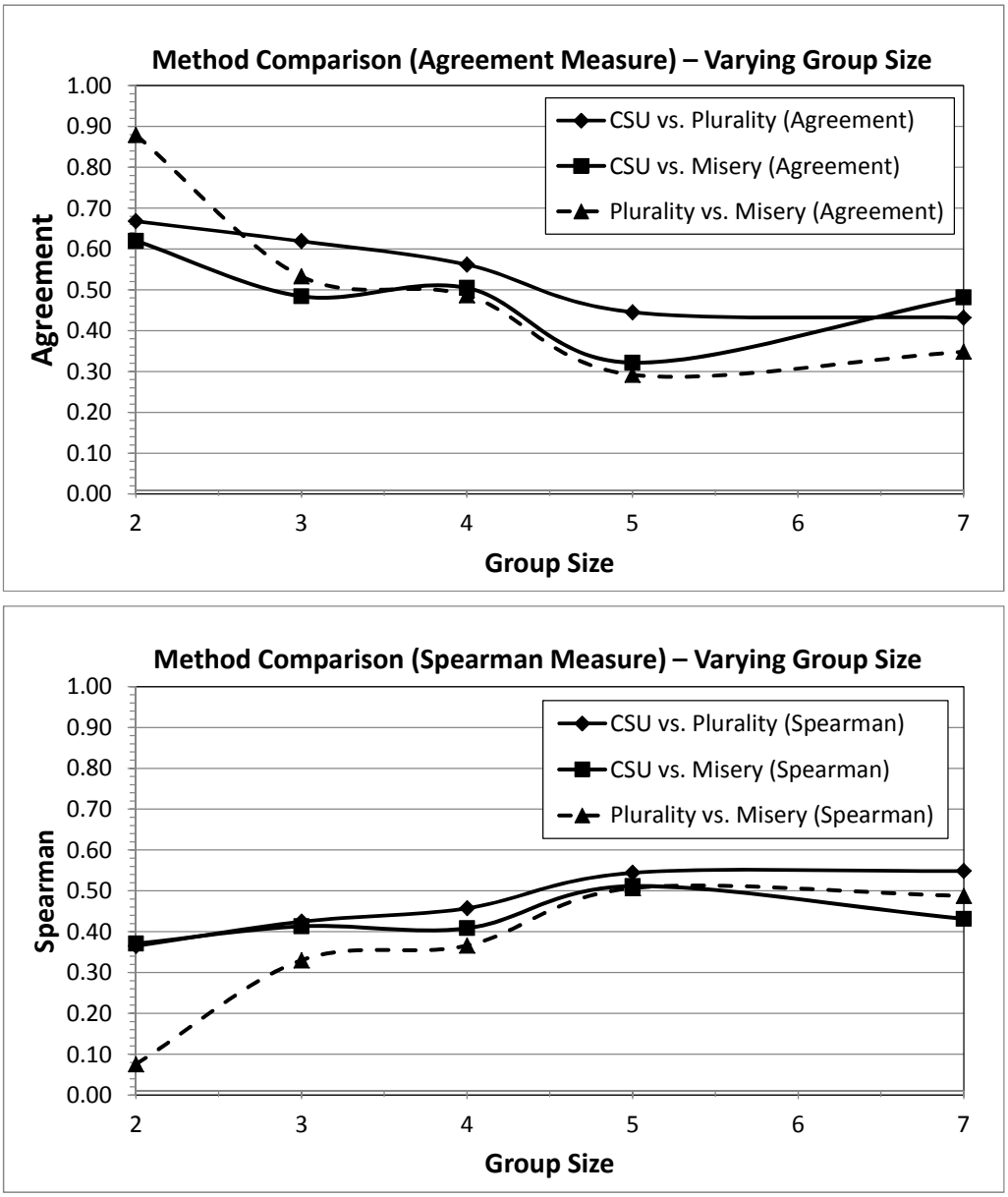


Figure 4.20: How Agreement (left; higher is better) and Spearman (right; lower is better) change depending on group size.

Part III

Conditional Preferences for the Semantic Web

Chapter 5

Single User Conditional Preferences

The previous chapter has focused on answering disjunctions of atomic queries for a very general preference representation language, since answering conjunctive queries is intractable in this context [115]. In this chapter, the focus is on a less expressive language, CP-theories [50], but the resulting formalism has tractable conjunctive query answering for some if its fragments.

CP-theories (introduced in Section 2.2.2) are a representation language that can express qualitative and conditional preferences in a natural, concise, and flexible way (e.g., "if a property is an apartment, then I prefer to have parking included to having a gym, independent of its location").

The chapter first introduces ontological CP-theories, which are a novel combination of Datalog$^{\pm}$ ontologies with CP-theories. It defines CP-theory–based conjunctive queries, and their skyline and k-rank answers on top of ontological CP-theories. It analyses the computational complexity of deciding consistency, dominance, and CQ skyline membership for OCP-theories, providing (generic and concrete) precise complexity results for different types of complexity. It also provides several tractability results in the data complexity for the case where query answering in the underlying

classical ontology is tractable in the data complexity.

The integration between the ontology and the CP-theory is tight: on the one hand, CP-theory outcomes are constrained by the ontology, and, on the other hand, they directly inform how answers to CQs are ranked.

The rest of this chapter is organised as follows. Section 5.1 introduces ontological CP-theories, where preferences are expressed over ground atoms from a Datalog$^{\pm}$ ontology, as well as the syntax and the semantics of conjunctive queries (CQs) based on such CP-theories. Section 5.2 describes how to compute k-rank answers to CP-theory-based CQs, and provide general complexity and tractability results. Finally, Section 5.4 summarises the main results of this chapter.

5.1 Formalism

Let us first introduce ontological CP-theories (or OCP-theories), which extend CP-theories by ontologies and generate a list of ordered (or ranked) answers to a given conjunctive query based on preferences. They informally define preferences between conjunctions of atoms relative to an ontology. W.l.o.g., the set Δ_N of nulls is the set of all ground terms constructed from the set Δ of constants, and a set $\mathcal{F}$ of functions used to skolemize all existential variables in TGDs.

Definition 5.1.1. Let O be a Datalog$^{\pm}$ ontology O over Δ. Let $\mathcal{X}$ be a finite set of variables, each $X \in \mathcal{X}$ being associated with a predicate p from O, denoted $p(X)$, and as domain $Dom(X)$ with a finite set of at least two different ground atoms $p(c_1, \ldots, c_k)$, with $c_1, \ldots, c_k \in \Delta \cup \Delta_N$. Let $Dom^+(X)$ be the set of all (non-ground) atoms $p(t_1, \ldots, t_k)$ with terms $t_1, \ldots, t_k$ over Δ, $\mathcal{V}$, and $\mathcal{F}$. Then, an *ontological conditional preference* over $\mathcal{X}$ has the form

$$\upsilon \colon \xi \succ \xi' \, [W], \tag{5.1}$$

where (i) $\upsilon \in Dom^+(U)$ for some $U \subseteq \mathcal{X}$, (ii) $\xi, \xi' \in Dom^+(X)$ for some $X \in \mathcal{X} \setminus U$, and (iii) $W \subseteq \mathcal{X} \setminus (U \cup \{X\})$. We say $\upsilon\theta \colon \xi\theta \succ \xi'\theta\,[W]$, where θ is a substitution, is a *ground instance* as in Equation (5.1), if $\upsilon\theta \in Dom(U)$ and $\xi\theta, \xi'\theta \in Dom(X)$. Let Γ be a finite set of preferences as in Equation (5.1). Then, (O, Γ) is an *ontological CP-theory* (or *OCP-theory*).

Note that it is assumed a different predicate for each $\mathcal{X}$ (the variables are over different predicates of the atoms). The following example illustrates OCP-theories.

Example 5.1.1. An OCP-theory (O, Γ) is given by the ontology O of Example 2.1.1, and the following set Γ of ontological conditional preferences:

$$
\begin{aligned}
\Gamma = \{ & \top \colon \mathit{property}(X, \mathit{house}, L, C) \succ \mathit{property}(Y, \mathit{apt}, L, C)[\emptyset], \\
& \top \colon \mathit{propertyFeature}(X, g) \succ \mathit{propertyFeature}(Y, p)[\{T, L\}], \\
& \mathit{propertyFeature}(X, g) \colon \mathit{location}(l_1) \succ \mathit{location}(l_2)[\emptyset], \\
& \mathit{property}(X, \mathit{apt}, L, C) \wedge \mathit{propertyFeature}(X, p) \colon \mathit{location}(l_1) \succ \mathit{location}(l_2)[\emptyset], \\
& \mathit{property}(X, \mathit{house}, L, C) \colon \mathit{location}(l_1) \succ \mathit{location}(l_2)[\emptyset]\}
\end{aligned}
$$

Γ encodes the preferences of Bob. Bob would like to rent a property and he prefers a *house* over an *apartment* (statement (1) below), even though he can live also in the latter. Since he owns a car, he would always prefer a property with garage over one with pool irrespective of the property type or location. Bob knows that location l_1 has better gyms than l_2, and l_1 is further from her work than l_2. Therefore, if a property has a garage (g), then he would prefer the location l_1 over l_2, since he can commute by car. Still, if there is an apartment that has swimming pool (p), he would prefer location l_2 over l_1 (4). If Bob finds a house, he would prefer to be in location l_1 over l_2, since he is thinking to build her gym for training.

Based on the above description of Bob's preferences, let us introduce three variables T_O, L_O, and F_O with the predicates $p(T_O) = \mathit{property}, p(L_O) = \mathit{location}$, and $p(F_O) = \mathit{propertyFeature}$, and domains $Dom(T_O) = \{\mathit{property}(t_1), \mathit{property}(t_2),$

$property(t_3), property(t_4), property(t_5), property(t_6)\}$,
$Dom(L_O)$=$\{location(l_1)$, $location(l_2), location(l_3)\}$, and $Dom(F_O) = \{property-Feature(t_{10})$, $propertyFeature(t_{11})$, $propertyFeature(t_{12}), propertyFeature(t_{13})\}$
(Note that $property(t_1)$ and $\mathbf{t}_1$, both stand for $property(p_1, apt, l_1, e)$, see Table 2.1), respectively. An example of the outcome is o with $o(T_o) = \mathbf{t}_1$, $o(L_o) = location(l_1)$ and $o(F_o) = \mathbf{t}_{10}$, interpreted as the conjunction $property(t_1) \wedge location(l_1) \wedge propertyFeature(t_{10})$. ∎

Observe that every outcome o of an OCP-theory actually represents a conjunction of ground atoms over $\Delta \cup \Delta_N$. As a consequence of the underlying ontology, some of these outcomes may be inconsistent, and some other outcomes may be equivalent. We thus have to ensure that the preference relation encoded in an OCP-theory is well-defined, which is expressed in the notion of consistency of an OCP-theory. To define it, we need some preparatory definitions as follows.

5.1.1 Consistency and Dominance

Intuitively, consistency requires that a theory does not have any contradiction. To give a specific characterisation of a consistent CP-theory, let us first begin with discussing the consistency and ontological equivalence of an outcome.

Definition 5.1.2. An outcome o of (O, Γ) is *consistent* if the following ontology $O_o = O \cup \{o(X) \mid X \in \mathcal{X}\}$ is consistent.

Example 5.1.2. Continuing with the Example 5.1.1, the outcome o of (O, Γ) is consistent. Let (O', Γ) be an OCP-theory, with Γ from Example 5.1.1, $O' = (D', \Sigma')$, D from Table 2.1, and $\Sigma' = \{\sigma'\}$,

$$\sigma' = property(I, T, L, l) \wedge propertyFeature(I, g) \rightarrow \bot,$$

then the outcome o', with $o'(F_O) = propertyFeature(p_2, g)$, $o'(L_O) = location(l_2)$ and $o'(T_O) = property(p_2, house, l_2, l)$, is inconsistent, since it does not satisfy σ'

Note that given an OCP-theory (O, Γ), if the ontology O is inconsistent, then every outcome will be inconsistent. For example, if we add $friend(b, b)$ to the ontology O, then the ontology becomes inconsistent, since it fails the dependency $friend(P, P) \rightarrow \bot$. ∎

Definition 5.1.3. Two outcomes o and o' of (O, Γ) are *equivalent*, denoted $o \sim o'$, if the ontologies O_o and $O_{o'}$ have the same models, with $O_o = O \cup \{o(X) \mid X \in \mathcal{X}\}$ and $O_{o'} = O \cup \{o'(X) \mid X \in \mathcal{X}\}$. The equivalence class of an outcome o in $Dom(\mathcal{X})$, denoted $[o]_\sim$ is the set $\{o' \in Dom(\mathcal{X}) \mid o' \sim o\}$.

Example 5.1.3. Two inconsistent outcomes are always ontologically equivalent, as they both have no models. ∎

Definition 5.1.4. An *interpretation* π for (O, Γ) is a total order over the set of equivalence classes of the consistent outcomes of (O, Γ).

Let us now define the notion of a model of an OCP-theory.

Definition 5.1.5. For the conditional preference $\varphi = \xi \succ \xi' [W]$, let $\varphi^* = \{([o]_\sim, [o']_\sim) \mid o = \upsilon\theta \wedge \xi\theta \wedge \tau\theta$, $o' = \upsilon\theta \wedge \xi'\theta \wedge \tau'\theta$, $\tau, \tau' \in Dom^+(W)$, and θ is a substitution of a ground instance of $\varphi\}$. We say π *satisfies* (or *is a model of*) a ground conditional preference φ (Equation (5.1)), denoted $\pi \models \varphi$, if $\pi \supseteq \varphi^\star$. We say π *satisfies* (or *is a model of*) a set of conditional preferences Γ, denoted $\pi \models \Gamma$, if it satisfies all ground instances of $\varphi \in \Gamma$. We say π *satisfies* (or *is a model of*) an OCP-theory (O, Γ), denoted $\pi \models (O, \Gamma)$, if it satisfies both O and Γ.

Let us now define the notion of consistency of OCP-theories as the existence of a model as follows.

Definition 5.1.6. An OCP-theory (O, Γ) is *consistent* if it has a model.

Let us give an alternative characterisation of this notion of consistency. Given an OCP-theory (O, Γ), let $\Gamma^\star_\sim$ denote the transitive closure of the set of all $([o]_\sim, [o']_\sim)$

such that (i) $([o]_\sim, [o']_\sim) \in \Gamma^\star$, and (ii) o and o' are both consistent. Then, the consistency of OCP-theories describes the acyclicity of $\Gamma^\star_\sim$.

Theorem 5.1.1. An OCP-theory (O, Γ) is consistent iff $\Gamma^\star_\sim$ is acyclic.

Proof. $\Rightarrow$ Assume (O, Γ) is consistent, that is, it has a model, then a total order of the equivalence classes of consistent outcomes exists. The total order contains order pairs. The transitive closure of the total order will contain the transitive closure of the order pairs. Since the transitive closure of total closure is acyclic, then the transitive closure of the order pairs will be acyclic, thus $\Gamma^\star_\sim$ will also be acyclic.

$\Leftarrow$ Assume $\Gamma^\star_\sim$ is acyclic, then there are no two nonequivalent consistent outcomes o, o' such that $(o, o') \in \Gamma^\star$ and $(o', o) \in \Gamma^\star$. We can build a model from $\Gamma^\star_\sim$, which proves that (O, Γ) is consistent.

□

The notions of dominance between consistent outcomes and of undominance of consistent outcomes are as follows.

Definition 5.1.7. Let (O, Γ) be a consistent OCP-theory, and o and o' be consistent outcomes. Then, o *dominates* o' in (O, Γ), denoted $o \succ o'$, if $([o]_\sim, [o']_\sim) \in \pi$ for all models π of (O, Γ). A consistent outcome o is *undominated* in (O, Γ), if no consistent outcome o' exists with $o' \succ o$.

The next theorem shows that the notion of dominance between consistent outcomes is exactly expressed by $\Gamma^\star_\sim$.

Theorem 5.1.2. Let (O, Γ) be a consistent OCP-theory, and o and o' be consistent outcomes. Then, $o \succ o'$ iff $([o]_\sim, [o']_\sim) \in \Gamma^\star_\sim$.

Proof. $\Rightarrow$ Assume $o \succ o'$. From Theorem 5.1.1, we know that $\Gamma^\star_\sim$ is acyclic. Then, we know that $[o]_\sim \succ [o']_\sim$, therefore $([o]_\sim, [o']_\sim) \in \Gamma^\star_\sim$.

$\Leftarrow$ Assume $([o]_\sim, [o']_\sim) \in \Gamma^\star_\sim$. From Theorem 5.1.1, we know that $\Gamma^\star_\sim$ is acyclic, therefore $o \nsim o'$, therefore $o \succ o'$.

□

5.2 Query Answering based on User Preferences

Now that the notions of consistency and dominance have been defined, let us elaborate on the semantics of query answering and move on to give definitions and algorithms for skyline and k-rank answering in the context of OCP-theories (O, Γ).

5.2.1 General Semantics for a Query

As a first step, let us formalise query answering for a given consistent OCP-theory (O, Γ).

Let $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$ be a query to the ontology O, where Φ is composed of atoms, and $\mathbf{X}, \mathbf{Y}$ are terms. Now, to extract answers based on the outcomes of CP-theories, the atoms in the query must be related to the atoms in preference statements. For this purpose, it is assumed a bijection β from a set of atoms $\Phi_\beta(\mathbf{X}, \mathbf{Y}) \subseteq \Phi(\mathbf{X}, \mathbf{Y})$ in Q to a set of variables of (O, Γ), such that for every atom $a \in \Phi_\beta(\mathbf{X}, \mathbf{Y})$ there exists some variable V in (O, Γ) so $pred(V) = a$ and $\beta(a) = V$. When β is empty, i.e., the query atoms are not related to preference atoms, then the answers for the query are unrelated to preferences, and thereby are standard CQ answers.

Definition 5.2.1. Let (O, Γ) be a consistent OCP-theory with $O = (D, \Sigma)$, and let $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ be a CQ where Φ is a conjunction of atoms. Let $o \in Dom(\mathcal{X})$ be a consistent outcome of (O, Γ). The set of all *answers of Q under the consistent outcome o*, denoted $ans(Q, O, \Gamma, o)$, is the set of all tuples $\mathbf{a}$ over $\Delta \cup \Delta_N$, for which a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ exists such that (i) $D \cup \Sigma \models h(\Phi(\mathbf{X}, \mathbf{Y}))$, (ii) $h(\mathbf{X}) = \mathbf{a}$, and (iii) $h(a) = o(\beta(a))$ for all $a \in \Phi_\beta(\mathbf{X}, \mathbf{Y})$.

Indeed, we may have more than one undominated consistent outcome o, and also more than one homomorphism h that satisfies the conditions (i) and (ii) of Definition 5.2.1 for the same undominated consistent outcome o.

Definition 5.2.2. An *answer* for a query Q to an OCP-theory (O, Γ) is an answer for Q to (O, Γ) under some consistent outcome o of (O, Γ).

As a next step one needs to order these answers based on the preferences of the user. I do this via skyline answers [16], a well-known class of answers for preference-based formalisms, and the iterated computation of skyline answers that allows us to assign a *rank* to every answer (via the CP-theory); we refer to these as k-rank answers.

Let us now first define skyline answers; note that they are not unique.

Definition 5.2.3. Let (O, Γ) be a consistent OCP-theory, and let Q be a query. A *skyline answer* for Q to (O, Γ) is any tuple $\mathbf{a} \in ans(Q, O, \Gamma, o)$ for some consistent outcome o of (O, Γ) such that there exists no consistent outcome o' with $o' \succ o$ and $ans(Q, O, \Gamma, o') \neq \emptyset$.

Let us next define k-rank answers for queries to OCP-theories via iteratively applying the notion of skyline answer. The definition is technically involved, as answers are obtained by projecting away existentially quantified arguments, thus distinct outcomes may be associated with the same answer, and so distinct levels of the skyline iteration may contain the same answers. This is handled in each iteration step by removing all answers computed in previous iteration steps.

Note that when no answer for q to (O, Γ) exists, then $\langle\rangle$ is its unique k-rank answer. Informally, a k-rank answer is a sequence of k answers for a CQ to an OCP-theory ranked by following the order among consistent outcomes induced by the OCP-theory. Clearly, as skyline answers are in general not unique, also k-rank answers are in general not unique.

Definition 5.2.4. Let (O, Γ) be a consistent OCP-theory and let Q be a query. A *k-rank answer* for Q to (O, Γ) outside a set of given ground atoms S is a sequence $\langle a_1, \ldots, a_k \rangle$ such that either:

(a) $\mathbf{a_1}, \ldots, \mathbf{a_k}$ are k different skyline answers for q to (O, Γ) not belonging to S, if at least k such answers exist; or

(b) (1) $\mathbf{a_1}, \ldots, \mathbf{a_i}$ are all i different skyline answers for q to (O, Γ) that do not belong to S, and (2) $\langle \mathbf{a_{i+1}}, \ldots, \mathbf{a_k} \rangle$ is a $(k-i)$-rank answer for q to $(O, \Gamma \backslash \{o\})$ outside $S \cup \{\mathbf{a_1}, \ldots, \mathbf{a_i}\}$, where o is an undominated outcome relative to $\succ$, otherwise.

A *k-rank answer* for q to (O, Γ) is a k-rank answer for q to (O, Γ) outside $\emptyset$.

The following example demonstrates the query answering for CQs. Note that a query may have multiple answers under the same outcome. The following example has a query with multiple homomorphisms, and thus multiple answers, under the same outcome satisfying conditions (i) and (ii) in the definition of CQs.

Example 5.2.1. Consider the consistent OCP-theory (O, Γ) from Example 5.1.1 and the CQ $Q(P, F) = \exists L\, property(P, apt, L, e) \wedge propertyFeature(P, F)$. An answer under the consistent outcome o with $o(T_o) = property(t_1)$, $o(F_o) = propertyFeature(t_{10})$ and $o(L_o) = location(l_1)$ is $\langle p_1, g \rangle$. On the other hand, under another consistent outcome o' with $o'(T_O) = property(t_1)$ and $o'(F_O) = propertyFeature(t_{11})$, the set of answers is empty, as two different constants p_1 and p_2 occur in $property$ and $propertyFeature$ in this outcome. Thus, the CQ is not satisfied.

Taking another CQ, $Q'(U') = \exists P, F\ propertyFeature(P, F) \wedge offer(U, P) \wedge friend(U, U')$. Under the outcome o, $o(T_o) = property(t_2)$, $o(F_o) = propertyFeature(t_{11})$ and $o(L_o) = location(l_1)$, we have the following set of answers: $\{\langle a \rangle, \langle m \rangle\}$ from different homomorphisms. ∎

5.2.2 Algorithms for Query Answering

In this section, I illustrate some algorithms pertaining to various computational problems in the context of preference-based query answering using OCP-theories, like computing an interpretation of a given consistent OCP-theory, query answering and ranking the answers. I also examine the sources of nondeterminism in the algorithms, which will be useful in understanding the computational complexity.

Answers for a CQ under an outcome. Algorithm 9 generates all the answers to a CQ for an OCP-theory under a given consistent outcome. It takes a CQ, OCP-theory and a consistent outcome as an input following Definition 5.2.1. There is one source of nondeterminism—in line 3, when one arbitrarily chooses a homomorphism h to generate a list of answers.

Algorithm 9 Answers for a CQ under an outcome

Input: (i) consistent OCP-theory (O, Γ),
(ii) CQ $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$, and
(iii) some consistent outcome o.
Output: list of answers $\langle a_1, \ldots, a_i \rangle$ for Q to (O, Γ) under o.
Called by: Algorithm 10.

```
1: procedure CQAns((O, Γ), Q, o)
2:     answers ← ∅;
3:     for all h such that O ⊨ h(Φ(X, Y)) do
4:         if h(a) = o(β(a)) for all a ∈ Φ_β(X, Y) then
5:             if h(X) ∉ answer then
6:                 answers ← answers · ⟨h(X)⟩;
7:     return answer.
```

k-rank answers for a query. Given an OCP-theory, a query and an integer $k \geq 0$ it outputs a list of answers. It selects an undominated outcome o arbitrarily. Based on o, it generates a list of answers (using Algorithm 9), and adds them to *result*, if not already there, until the length of the becomes k. At this point, it also removes from *alloutcomes* any order which is of the form (o, o'), where o is the currently chosen outcome, so that we do not encounter the same outcome again. If the length is still less than k, then it selects another outcome arbitrarily from *outcomes*, and

proceeds similarly. There are two sources of nondeterminism—in line 9, when we choose an arbitrary outcome, and in line 13, when any answer is chosen arbitrarily until k is reached (it may so happen that for some outcome, the algorithm reaches k before we exhaust all answers corresponding to it). Also, to compute the set of all skyline answers, we remove lines 5, 16, 17 and all statements and Boolean conditions where *k-reached* occurs.

Algorithm 10 k-rank answers for a query

Input: (i) consistent OCP-theory (O, Γ),
(ii)CQ Q and
(iii) $k \geq 0$.
Output: k-rank answers $\langle a_1, \ldots, a_i \rangle$ for Q to (O, Γ).
Calls: Algorithm 9.

```
1: procedure kRankPrefs((O, Γ), Q, k)
2:     result ← ∅;
3:     checked ← ∅;
4:     outcomes ← ∅;
5:     k-reached ← false;
6:     while k-reached = false do
7:         for all undominated outcomes o relative to ≻ by removing all pairs
       o1 ≻ o2 such that o ∉ checked do
8:             outcomes ← outcomes ∪ {o};
9:         while k-reached = false and outcomes ≠ ∅ do
10:            o = some outcome ∈ outcomes;
11:            outcomes ← outcomes − {o};
12:            checked ← checked ∪ {o};
13:            answers ← CQAns(O, Γ, Q, o); // Algorithm 9
14:            for all a ∈ answers do
15:                if k-reached= false and a ∉ result then
16:                    result ← result · ⟨a⟩;
17:                    if length(result) = k then
18:                        k-reached ← true;
19:    return result.
```

5.3 Computational Complexity

I now analyse the computational complexity of deciding consistency, dominance, and CQ skyline membership for OCP-theories. I also delineate some tractable special cases. Since to compute a set of skyline answers and k-rank answers it requires a

polynomial-time call to CQ skyline membership, the same complexity results will hold for these problems as well.

I assume some familiarity with the complexity classes PSPACE (resp.,P, EXP, 2EXP) containing all decision problems that can be solved in polynomial space (resp., polynomial, exponential, double exponential time) on a deterministic Turing machine; see [87, 136]. In the following, BCQ$^{\text{N}}$s denote BCQs with elements from $\Delta \cup \Delta_N \cup \mathcal{V}$ as arguments in atoms.

5.3.1 Combined Complexity

Recalling from Section 2.1.2, in this section, I analyse the combined *ba*-combined and fp-combined complexity of various decision problems associated with OCP-theories. Before talking about specific languages, I derive general results.

For complexity classes $\mathcal{C} \subseteq$ PSPACE, the following theorem gives the generic result for (i) deciding consistency of OCP-theories (O, Γ) and (ii) deciding dominance between two outcomes of OCP-theories (O, Γ), given that BCQ answering in O is complete for $\mathcal{C}$ and hardness holds even for ground atomic BCQs.

Theorem 5.3.1. Let $\mathcal{T}$ be a class of OCP-theories (O, Γ) where BCQ^N answering in O is complete for a complexity class $\mathcal{C} \subseteq$ PSPACE, and hardness holds even for ground atomic BCQs. Then, deciding whether

(a) some given $(O, \Gamma) \in \mathcal{T}$ is consistent, and
(b) o dominates o' in (O, Γ), given $(O, \Gamma) \in \mathcal{T}$ and two outcomes o, o',

are both complete for PSPACE.

Proof.

(a) **Membership.** By Theorem 5.1.1, to decide consistency, it is sufficient to decide whether $\Gamma^\star_\sim$ is acyclic. That means, I need to decide if there is a consistent outcome that dominates itself (i.e., decide if $[o]_\sim \succ [o]_\sim$ or $([o]_\sim, [o]_\sim) \in \Gamma^\star_\sim$ for some consistent outcome o). The number of outcomes of (O, Γ) is exponential,

but it can be done by storing at most two outcomes, while exploring all the paths in $\Gamma^{\star}_{\sim}$. At each step, we would check the consistency of the outcomes and their equivalence. These two checks are equivalent with deciding BCQ^{N}s to O, which is in $\mathcal{C}$. A we saw in Section 2.2.2.1, to decide the dominance between two outcomes in a standard CP-theory is in PSPACE [72]. Thus, overall, this is possible in PSPACE, since we assumed $\mathcal{C} \subseteq$ PSPACE.

Hardness. PSPACE-hardness follows from the more specialised problem of deciding consistency in a standard CP-theory being PSPACE-hard [72].

Let Γ be a CP-theory over $\mathcal{X}$. We construct (O, Γ') an OCP-theory over $\mathcal{X}_O$ such that for each $X_i \in \mathcal{X}$ (from the CP-theory), we construct a predicate p_i and the corresponding OCP-theory variable X'_i, with $Dom(X'_i) = \{p_i(x) | x \in Dom(X_i)\}$.

We construct the ontology $O = (\Sigma, D)$, with $\Sigma = \emptyset$. The relational schema contains all the created predicates of unary arity, that is, $\mathcal{R} = \{p_i | X_i \in \mathcal{X}\}$. The atoms of D for $\mathcal{R}$ are all the atoms from $Dom(\mathcal{X}_O)$.

For each conditional preference φ (of the form $v\colon \xi \succ \xi' [W]$ with $\xi, \xi' \in Dom(X_i)) \in \Gamma$ over $\mathcal{X}$, we construct an ontological conditional preference $\varphi' \in \Gamma'$ over $\mathcal{X}_O$ of the form $v'\colon p_i(\xi) \succ p_i(\xi') [W']$, where $v' = \{\bigwedge p_j(y_j) | Y_j \in U, y_j \in v[Y_j]\}$ (the value of Y_j is y_j) and $W' = \{Y'_j | Y_j \in W\}$.

We now show that Γ is consistent iff (O', Γ') is consistent.

$\Rightarrow$ Suppose Γ is consistent. This means that Γ has a model (that is, there is a total order π of the outcomes of Γ). We will construct a model π_O of (O, Γ'). For each pair of outcomes $(o, o') \in \pi$ with $o = o_1 \dots o_n$ and $o' = o'_1 \dots o'_n$, we add (o_O, o'_O) to π_O such that $o_O = \bigwedge_{i=1}^{n} p_i(o_i)$ and $o'_O = \bigwedge_{i=1}^{n} p_i(o'_i)$. Since Σ has no rules, we do not have equivalent different outcomes.

⇐ Suppose (O, Γ') is consistent. This means that (O, Γ') has a model (that is, there is a total order π_O of the equivalence classes of consistent outcomes of (O, Γ')). We will construct a model π of Γ. For each pair of outcomes $([o_O]_\sim, [o'_O]_\sim) \in \pi_O$ with $o_O = \bigwedge_{i=1}^{n} p_i(o_i)$ and $o'_O = \bigwedge_{i=1}^{n} p_i(o'_i)$, we add (o, o') to π such that $o = o_1 \dots o_n$ and $o' = o'_1 \dots o'_n$, because since $\Sigma = \emptyset$, we do not have equivalent different outcomes.

(b) **Membership.** As for membership, by Theorem 5.1.2, it is sufficient to decide whether o and o' are consistent and whether $[o]_\sim \succ [o']_\sim$ ($([o]_\sim, [o']_\sim) \in \Gamma^\star_\sim$). As argued in (a), this is overall possible in PSPACE.

Hardness. PSPACE-hardness follows from the PSPACE-hardness of the more specialised problem of deciding dominance between two outcomes in standard CP-theories [72]. We will use the transformation that we used in (a). Given o and o' two outcomes of Γ with $o = o_1 \dots o_n$ and $o' = o'_1 \dots o'_n$ and $o_O = \bigwedge_{i=1}^{n} p_i(o_i)$, $o'_O = \bigwedge_{i=1}^{n} p_i(o'_i)$.

We now show that o dominates o' in Γ iff o_O dominates o'_O in (O, Γ')

⇒ Suppose o dominates o' in Γ. Since we have a direct mapping between the outcomes of Γ and the outcome of (O, Γ'), it follows $o_O \succ o'_O$.

⇐ Suppose o_O dominates o'_O in (O, Γ'). Since we have a direct mapping between the outcomes of Γ and the outcome of (O, Γ'), it follows $o \succ o'$

□

The next theorem shows the generic results that (i) deciding consistency of OCP-theories (O, Γ) and (ii) deciding dominance between two outcomes of OCP-theories (O, Γ) are both complete for $\mathcal{C}$ when BCQ^N answering in O is complete for the deterministic complexity class $\mathcal{C} \supseteq$ EXP, and hardness holds even for ground atomic BCQs.

Theorem 5.3.2. Let $\mathcal{T}$ be a class of OCP-theories (O, Γ) where BCQ^N answering in O is complete for a deterministic complexity class $\mathcal{C} \supseteq$ EXP, and hardness holds even for ground atomic BCQs. Then, deciding whether

(a) some given $(O, \Gamma) \in \mathcal{T}$ is consistent, and
(b) o dominates o' in (O, Γ), given $(O, \Gamma) \in \mathcal{T}$ and two outcomes o, o',

are both complete for $\mathcal{C}$.

Proof.

(a) **Membership.** By Theorem 5.1.1, it is sufficient to decide whether $\Gamma^\star_\sim$ is acyclic. This can be done by deciding whether $[o]_\sim \succ [o]_\sim$ for some consistent outcome o, which can be done (despite the number of outcomes of (O, Γ) being exponential) by storing at most two outcomes, while exploring all the paths in $\Gamma^\star_\sim$. This requires both deciding BCQ^{N}s to O (for deciding if an outcome is consistent, and whether two consistent outcomes are equivalent), which is in $\mathcal{C}$, and deciding dominance between two outcomes in a standard CP-theory, which is in PSPACE [72]. Overall, this is possible in $\mathcal{C}$, since $\mathcal{C} \supseteq$ EXP$\supseteq$ PSPACE.

Hardness. Hardness for $\mathcal{C}$ follows from a reduction of the $\mathcal{C}$-hard problem of answering ground atomic BCQs.

Let O be an ontology and $\mathbf{a}$ be a ground atomic query.

We construct the OCP-theory (O', Γ') over $\mathcal{X}_O = \{B\}$ with $Dom(B) = \{\mathbf{b}, \mathbf{b}'\}$, $\Gamma' = \{\mathbf{b}' \succ \mathbf{b}\}$ and $O' = O \cup \{\mathbf{b}', \mathbf{a} \rightarrow \mathbf{b}\}$. That is, we have that $\mathbf{b}$ and $\mathbf{b}'$ are two outcomes for (O', Γ'), since we have only one variable B, whose domain is two ground atoms, $\mathbf{b}$ and $\mathbf{b}'$. Note that $\mathbf{b}$ and $\mathbf{b}'$ are two fresh ground atoms that are not part of O.

We now show that $O \models \mathbf{a}$ iff (O', Γ') is inconsistent.

$\Rightarrow$ Suppose $O \models \mathbf{a}$. We now prove that $\mathbf{b}$ and $\mathbf{b}'$ are equivalent outcomes of (O', Γ') and by having $\mathbf{b}' \succ \mathbf{b}$ in Γ', we have that (O', Γ') is inconsistent.

From $O \models \mathbf{a}$, we can deduce that $O' \models \mathbf{b}$, since $\mathbf{a} \rightarrow \mathbf{b}$. Using the definition $O' \cup \{\mathbf{b}(B)\} = O \cup \{\mathbf{b}, \mathbf{b}', \mathbf{a} \rightarrow \mathbf{b}\}$ and $O' \cup \{\mathbf{b}'(B)\} = O \cup \{\mathbf{b}', \mathbf{a} \rightarrow \mathbf{b}\} = O \cup \{\mathbf{b}', \mathbf{b}, \mathbf{a} \rightarrow \mathbf{b}\}$. Since $O' \cup \{\mathbf{b}(B)\}$ and $O' \cup \{\mathbf{b}'(B)\}$ have the same model, we conclude that $\mathbf{b} \sim \mathbf{b}'$. Therefore, we have proven that $\mathbf{b}$ and $\mathbf{b}'$ are equivalent outcomes of (O', Γ') .

$\Leftarrow$ Suppose (O', Γ') is inconsistent. Then by Theorem 5.1.1, we have that $\Gamma'^{\star}_{\sim}$ is cyclic. Since Γ' contains only one variable, it holds $\mathbf{b} \sim \mathbf{b}'$. To prove the equivalence, we have to prove $O' \cup \{\mathbf{b}'\} \models O' \cup \{\mathbf{b}\}$ ($O' \cup \{\mathbf{b}\} \models O' \cup \{\mathbf{b}'\}$ by definition). What we need to prove is that $O' \cup \{\mathbf{b}'\} \models \mathbf{b}$, or $O' \models \mathbf{b}$ (since $O' \models \mathbf{b}'$). The only way this could be achieved is by having $O \models \mathbf{a}$.

Thus, we reduced the known $\mathcal{C}$-hard problem, namely answering ground atomic BCQs to the problem of consistency checking of an OCP-theory.

(b) **Membership.** As for membership, by Theorem 5.1.2, it is sufficient to decide whether o and o' are consistent and whether $[o]_{\sim} \succ [o']_{\sim}$. As argued in (a), this is overall possible in $\mathcal{C}$.

Hardness. Hardness for $\mathcal{C}$ holds by a reduction from the problem of answering ground atomic BCQs.

Let O be an ontology and $\mathbf{a}$ a ground atomic query.

We construct an OCP-theory (O', Γ') over $\mathcal{X}_O = \{B\}$ with $Dom(B) = \{\mathbf{b}, \mathbf{b}', \mathbf{b}''\}$, $\Gamma' = \{\mathbf{b} \succ \mathbf{b}'\}$ and $O' = O \cup \{\mathbf{b}'', \mathbf{a} \rightarrow \mathbf{b}\}$. Note that $\mathbf{b}$, $\mathbf{b}'$ and $\mathbf{b}''$ are outcomes, since we have only one variable in the given OCP-theory, and since they contain fresh ground atoms that are not part of O.

We now show that $O \models \mathbf{a}$ iff $\mathbf{b}'' \succ \mathbf{b}'$ holds in (O', Γ').

$\Rightarrow$ Suppose $O \models \mathbf{a}$. We now prove that $\mathbf{b}$ and $\mathbf{b}''$ are equivalent outcomes of (O', Γ') and by having $\mathbf{b} \succ \mathbf{b}'$ in Γ', we then also have $\mathbf{b}'' \succ \mathbf{b}'$.

By definition, $O' \cup \{\mathbf{b}(B)\} = O' \cup \{\mathbf{b}\} = O \cup \{\mathbf{b}, \mathbf{b}'', \mathbf{a} \rightarrow \mathbf{b}\}$ and $O' \cup \{\mathbf{b}''(B)\} = O \cup \{\mathbf{b}'', \mathbf{a} \rightarrow \mathbf{b}\} = O \cup \{\mathbf{b}'', \mathbf{ba} \rightarrow \mathbf{b}\}$ (since $O \models \mathbf{a}$, therefore $O' \models \mathbf{b}$, since $\mathbf{a} \rightarrow \mathbf{b}$). Since $O' \cup \{\mathbf{b}(B)\}$ have the same models $O' \cup \{\mathbf{b}''(B)\}$, we conclude that $\mathbf{b} \sim \mathbf{b}''$.

$\Leftarrow$ Suppose $\mathbf{b}'' \succ \mathbf{b}'$ holds in (O', Γ'). Then, by Theorem 5.1.2, we have that $\Gamma'^{\star}_{\sim}$ contains $([\mathbf{b}'']_{\sim}, [\mathbf{b}]_{\sim})$. Since Γ' contains only one variable, then we have $\mathbf{b} \sim \mathbf{b}''$. To prove the equivalence, we have to prove that $O' \cup \{\mathbf{b}''\} \models O' \cup \{\mathbf{b}\}$ ($O' \cup \{\mathbf{b}\} \models O' \cup \{\mathbf{b}''\}$ by definition). What we need to prove is that $O' \cup \{\mathbf{b}''\} \models \mathbf{b}$, or $O' \models \mathbf{b}$ (since $O' \models \mathbf{b}''$). The only way, this could be achieved is by having $O \models \mathbf{a}$.

Since ground atomic BCQ answering is $\mathcal{C}$-hard, dominance testing is $\mathcal{C}$-hard as well, and thus overall $\mathcal{C}$-complete.

□

As a corollary, by the known complexity results for Datalog$^{\pm}$ (see, e.g., [122]), deciding consistency and dominance for OCP-theories when the underlying ontology is in L, LF, AF, G, WG, S, F, GF, SF, WS, or WA, is complete for the complexity classes shown in Tables 5.1 to 5.3 in the combined, *ba*-combined, and *fp*-combined complexity.

Corollary 5.3.3. Given an OCP-theory (O, Γ), where O is defined in L, LF, AF, G, WG, S, F, GF, SF, WS, or WA, deciding whether (a) (O, Γ) is consistent and (b) o dominates o', given (O, Γ) and two outcomes o and o', is both complete for the complexity classes shown in Tables 5.1 to 5.3 in the combined, *ba*-combined, and *fp*-combined complexity.

We next show that the complexity of deciding whether a tuple over Δ is in the skyline answer for a CQ to an OCP-theory (O, Γ) is complete for PSPACE when BCQ^N answering in O is complete for the complexity class $\mathcal{C} \subseteq$ PSPACE.

Theorem 5.3.4. Let $\mathcal{T}$ be a class of consistent OCP-theories (O, Γ) where BCQ^N answering in O is complete for a complexity class $\mathcal{C} \subseteq$ PSPACE. Then, given $(O, \Gamma) \in \mathcal{T}$, a CQ Q, and a tuple $\mathbf{a}$ over $\Delta \cup \Delta_N$, deciding whether $\mathbf{a}$ is in the skyline answer for q to (O, Γ) is complete for PSPACE.

Proof. **Membership.** For membership, it is sufficient to decide if $\mathbf{a} \in ans(Q, O, \Gamma, o)$ for some consistent outcome o such that no consistent outcome o' exists with (i) $o' \succ o$ and (ii) $ans(Q, O, \Gamma, o') \neq \emptyset$. By a similar argument as in the proof of Theorem 5.3.1, this is overall possible in PSPACE.

Hardness. Hardness for PSPACE follows by a reduction from the PSPACE-hard problem of deciding dominance between two outcomes o and o in standard CP-theories.

Let Γ be a CP-theory over $\mathcal{X}$, and $o = o_1 \ldots o_n$ and $o' = o'_1 \ldots o'_n$ be two outcomes of Γ. We construct an OCP-theory (O, Γ') over $\mathcal{X}_O$ from the Γ. For each $X_i \in \mathcal{X}$ of Γ, we create a unary-predicate p_i and the corresponding OCP-theory variable X'_i, with $Dom(X'_i) = \{p_i(x) | x \in Dom(X_i)\}$, $X'_i \in \mathcal{X}_O$, $Dom(\mathcal{X}_O) = \{p_i(X'_i) | X'_i \in \mathcal{X}_O\}$. Let the corresponding outcome of o (resp., o') from Γ be the outcome o_O of (O, Γ') with $o_O = \bigwedge_{i=1}^{n} p_i(o_i)$ (resp., $o_O = \bigwedge_{i=1}^{n} p_i(o'_i)$).

We construct the ontology $O = (\Sigma, D)$, with $\Sigma = \{\top \rightarrow \bigwedge_{i=1}^{n} p_i(o_i);\ \top \rightarrow \bigwedge_{i=1}^{n} p_i(o'_i)\}$. The relational schema contains all the created predicates of unary arity, that is, $\mathcal{R} = \{p_i | X_i \in \mathcal{X}\}$, while D is an empty database. We construct the query $q(X_1, \ldots X_n) = \bigwedge_{i=1}^{n} p_i(X_i)$

We now show that $o \succ o'$ in Γ iff $(o_1, \ldots, o_n)$ is in the skyline answer for q to (O, Γ').

$\Rightarrow$ Suppose $o \succ o'$ in Γ, then $o_O \succ o'_O$ in (O, Γ') (since there is a direct bijection between the outcomes of the CP-theory and the outcomes of the OCP-theory). The only answers to q are o and o' ($o = ans(q, O, \Gamma', o_O)$ and $o' = ans(q, O, \Gamma', o'_O)$), since the only consistent outcomes are o_O and o'_O (according to Σ). Assume that o is not in the skyline answer, which means

that o' is in the skyline answer, $o'_O \succ o_O$, which is a contradiction. Therefore, o is in the skyline answer for q to (O, Γ').

$\Leftarrow$ Suppose o is in the skyline answer for q to (O, Γ') (the corresponding undominated outcome is o_O, $o = ans(q, O, \Gamma', o_O)$, since we have a bijection between the outcomes of the CP-theory and the outcomes of the OCP-theory), that is, no consistent outcome o''_O exists with (i) $o''_O \succ o_O$ and (ii) $ans(q, O, \Gamma', o''_O) \neq \emptyset$. Since we have only two consistent outcomes in (O, Γ') (o_O and o'_O), it follows that $o_O \succ o'_O$; therefore, $o \succ o'$.

Since dominance testing between two outcomes in standard CP-theories is PSPACE-hard, deciding the skyline answer is PSPACE-hard as well, and thus overall $\mathcal{C}$-complete. □

The following theorem shows that the complexity of deciding whether a tuple over Δ is in the skyline answer for a CQ to an OCP-theory (O, Γ) is complete for $\mathcal{C}$ when BCQ^N answering in O is complete for the deterministic complexity class $\mathcal{C} \supseteq$ EXP.

Theorem 5.3.5. Let $\mathcal{T}$ be a class of consistent OCP-theories (O, Γ) where BCQ^N answering in O is complete for a deterministic complexity class $\mathcal{C} \supseteq$ EXP. Then, given $(O, \Gamma) \in \mathcal{T}$, a CQ Q, and a tuple $\mathbf{a}$ over $\Delta \cup \Delta_N$, deciding whether $\mathbf{a}$ is in the skyline answer for q to (O, Γ) is complete for $\mathcal{C}$.

Proof.

Membership. Given a CQ Q, an answer $\mathbf{a}$ for an OCP-theory (O, Γ) is a tuple over Δ. Deciding membership in a skyline answer is equivalent to checking if $\mathbf{a} \in ans(Q, O, \Gamma, o)$ for some consistent outcome o such that no consistent outcome o' exists with (i) $o' \succ o$ and (ii) $ans(Q, O, \Gamma, o') \neq \emptyset$, by the definition of skyline answers (see Definition 5.2.3). Evaluating $ans(Q, O, \Gamma, o)$ for a CQ Q to an OCP-theory under a given outcome o is equivalent to evaluating a set of answers for Q in the

ontology O, which is $\mathcal{C}$-complete. Furthermore, deciding if there exists a consistent outcome o' that satisfies the conditions (i) and (ii) from above requires checking consistency and dominance between two outcomes and deciding BCQ answers for Q under outcome o'. By a similar argument as in the proof of Theorem 5.3.2, this is overall possible in $\mathcal{C}$ (BCQ answering for O). Thus, overall, the problem of deciding skyline membership lies in $\mathcal{C}$.

Hardness. Hardness for $\mathcal{C}$ holds by a reduction from the problem of answering ground atomic BCQs.

Let O be an ontology and $\mathbf{a}$ a ground atomic query.

We construct an OCP-theory (O', Γ') over $\mathcal{X}_O = \{B\}$ with $Dom(B) = \{p(b), p(b'), p(b'')\}$, $\Gamma' = \{p(b') \succ p(b)\}$ and $O' = O \cup \{p(b''), \mathbf{a} \rightarrow p(b)\}$. Note that $p(b)$, $p(b')$ and $p(b'')$ are outcomes, since we have only one variable in the given OCP-theory, and since they are fresh ground atoms that are not part of O (p is a new unary predicate that is not part of the relational schema of O).

We now show that $O \models \mathbf{a}$ iff b' is in the skyline answer of $p(X)$ in (O', Γ').

$\Rightarrow$ Suppose $O \models \mathbf{a}$. We now prove that $p(b)$ and $p(b'')$ are equivalent outcomes of (O', Γ'), and by having $p(b') \succ p(b)$ in Γ', we have $p(b') \succ p(b'')$ as well.

By definition, $O' \cup \{p(b)\} = O \cup \{p(b), p(b''), \mathbf{a} \rightarrow p(b)\}$ and $O' \cup \{p(b'')\} = O \cup \{p(b''), \mathbf{a} \rightarrow p(b)\} = O \cup \{p(b), p(b''), \mathbf{a} \rightarrow p(b)\}$ (since $O \models \mathbf{a}$ implies $O' \models p(b)$, because $\mathbf{a} \rightarrow p(b)$).

Since $O' \cup \{p(b)\}$ and $O' \cup \{p(b'')\}$ have the same models, we conclude that $p(b) \sim p(b'')$ for (O', Γ').

The answers of $p(X)$ are b, b', b''. Since $p(b') \succ p(b'')$ and $p(b') \succ p(b'')$, it follows that b' is in the skyline answer of $p(X)$ in (O', Γ').

$\Leftarrow$ Suppose b' is in the skyline answer of $p(X)$ in (O', Γ'). The answers of $p(X)$ are b, b', b'', therefore, $p(b') \not\succ p(b'')$ (we can have that $p(b'') \succ p(b')$).

To prove $p(b) \sim p(b'')$, we show that $O' \cup \{p(b'')\} \models O' \cup \{p(b)\}$ ($O' \cup \{p(b)\} \models$

Language	Data	Comb.	*ba*-comb.	*fp*-comb.
L	in P	PSPACE-C	PSPACE-C	PSPACE-C
LF	in P	PSPACE-C	PSPACE-C	PSPACE-C
AF	in P	PSPACE-C	PSPACE-C	PSPACE-C
G	in P	2EXP-C	EXP-C	PSPACE-C
WG	EXP-C	2EXP-C	EXP-C	EXP-C
S	in P	EXP-C	PSPACE-C	PSPACE-C
GF, SF	in P	EXP-C	PSPACE-C	PSPACE-C
F	in P	EXP-C	PSPACE-C	PSPACE-C
A	in P	NEXP-C	NEXP-C	PSPACE-C
WA	in P	2EXP-C	2EXP-C	PSPACE-C
WS, WSJ	in P	2EXP-C	2EXP-C	PSPACE-C
T	in P	2EXP-C	EXP-C	PSPACE-C

Table 5.1: Data, combined, *ba*-combined, and *fp*-combined complexity of deciding consistency, dominance, and CQ skyline membership for OCP-theories in different languages.

$O' \cup \{p(b'')\}$ by definition). What we need to prove is that $O' \cup \{p(b'')\} \models p(b)$, or $O' \models p(b)$ (since $O' \models p(b'')$). The only way this could be achieved is by having $O \models \mathbf{a}$.

Since ground atomic BCQ answering is $\mathcal{C}$-hard, dominance testing is $\mathcal{C}$-hard as well, and thus overall $\mathcal{C}$-complete. □

By the known complexity results for Datalog$^{\pm}$ (see, e.g., [122]), we obtain as a corollary that deciding CQ skyline membership for OCP-theories when the underlying ontology is in L, LF, AF, G, WG, S, F, GF, SF, WS, or WA, is complete for the complexity classes in Tables 5.1 to 5.3 in the combined, *ba*-combined, and *fp*-combined complexity.

Corollary 5.3.6. Given a consistent OCP-theory (O, Γ), where O is defined in L, LF, AF, G, WG, S, F, GF, SF, WS, or WA, a CQ, and a tuple $\mathbf{a}$ over $\Delta \cup \Delta_N$, deciding whether $\mathbf{a}$ is in the skyline answer for q to (O, Γ) is complete for the classes in Tables 5.1 to 5.3 in the combined, *ba*-combined, and *fp*-combined complexity.

Language	Data	Comb.	*ba*-comb.
Y1,YG	in P	2EXP-C	2EXP-C
WYG	EXP-C	2EXP-C	2EXP-C
gG, gYG	EXP-hard	3EXP-C	3EXP-C
ba − Y1	in P	EXP-C	EXP-C
ba − YG	in P	2EXP-C	EXP-C

Table 5.2: Data, combined, and *ba*-combined complexity of deciding consistency, dominance, and CQ skyline membership for OCP-theories in different languages.

Language	Data	Comb.
SJ	in P	EXP-C
JA	in P	2EXP-C
JYG, JG	EXP-C	2EXP-C
H	in P	EXP-C
WAGRD	in P	in 2EXP

Table 5.3: Data and combined complexity of deciding consistency, dominance, and CQ skyline membership for OCP-theories in different languages.

5.3.2 Data Complexity

Let us now delineate special cases where deciding consistency, dominance, and CQ skyline membership for OCP-theories are all tractable in the data complexity. The following result shows that deciding consistency and dominance for OCP-theories (O, Γ), with $O = (D,\ \Sigma)$, is data tractable when BCQ^N answering in O is data tractable. Here, data complexity means that Σ and the variables and preferences of Γ are both fixed, while D and the domains of Γ are part of the input.

Theorem 5.3.7. Let $\mathcal{T}$ be a class of OCP-theories (O, Γ) where BCQ^N answering in O is possible in polynomial time in the data complexity. Then, deciding whether

(a) some given $(O, \Gamma) \in \mathcal{T}$ is consistent, and

(b) o dominates o' in (O, Γ), given $(O, \Gamma) \in \mathcal{T}$ and two outcomes o and o',

are both possible in polynomial time in the data complexity.

Proof.

(a) In the data complexity, the number of all outcomes of (O, Γ) is polynomial in the number of domain values since Γ and Σ are fixed, and the domain values are derived from the database instance.

Since deciding BCQs in O is possible in data-polynomial time, deciding if an outcome is consistent, and whether two consistent outcomes are equivalent as well. Therefore, computing $\Gamma^{\star}_{\sim}$ and checking its acyclicity, which are both dependent on outcome consistency and dominance, are also possible in data polynomial time. Overall, by Theorem 5.1.1, one knows that an OCP-theory is consistent iff $\Gamma^{\star}_{\sim}$ is acyclic. Therefore, deciding consistency of an OCP-theory (O, Γ) is in PTIME in the data complexity.

(b) By Theorem 5.1.2, it is sufficient to decide whether o and o' are consistent, and whether $([o]_{\sim}, [o']_{\sim}) \in \Gamma^{\star}_{\sim}$, which is possible in data-polynomial time. Thus, dominance testing between two outcomes of an OCP-theory (O, Γ) is also in PTIME in the data complexity.

□

As a corollary, deciding consistency and dominance for OCP-theories is data tractable when the underlying ontology is formulated in L, LF, AF, G, S, F, GF, SF, WS, or WA, which all allow for data-tractable BCQ^N answering.

Corollary 5.3.8. Given an OCP-theory (O, Γ), where O is defined in L, LF, AF, G, S, F, GF, SF, WS, or WA, deciding whether (a) (O, Γ) is consistent and (b) o dominates o', given (O, Γ) and outcomes o and o', are all possible in polynomial time in the data complexity.

The next result shows that deciding membership to skyline answers for CQs to consistent OCP-theories is data tractable, when BCQ^N answering in O is data-tractable.

Theorem 5.3.9. Let $\mathcal{T}$ be a class of consistent OCP-theories (O, Γ) where BCQ^N answering in O can be done in polynomial time in the data complexity. Then, given $(O, \Gamma) \in \mathcal{T}$, a CQ Q, and a tuple $\mathbf{a}$ over $\Delta \cup \Delta_N$, deciding whether $\mathbf{a}$ is in the skyline answer for q to (O, Γ) is possible in polynomial time in the data complexity.

Proof. As argued in the proof of Theorem 5.3.7, computing $\Gamma^{\star}_{\sim}$ can be done in data-polynomial time. Hence, by Theorem 5.1.2, computing iteratively undominated outcomes can also be done in data-polynomial time.

To decide skyline membership, one needs to check whether $\mathbf{a} \in ans(Q, O, \Gamma, o)$ for a CQ Q in an OCP-theory (O, Γ) under a given consistent outcome o, and if there exists an outcome o' such that (i) $o' \succ o$ and (ii) $ans(Q, O, \Gamma, o') \neq \emptyset$, both of which can be done in data-polynomial time, as argued before and by the fact that BCQ$^{\mathrm{N}}$ answering can be done in data-polynomial time. Thus, deciding whether an answer is a skyline answer is a polynomial-time problem in data complexity. □

A corollary is that membership to skyline answers for CQs to consistent OCP-theories is data-tractable when the ontology is formulated in L, LF, AF, G, S, F, GF, SF, WS, or WA, which all allow for data-tractable BCQ^N answering.

Corollary 5.3.10. Given an OCP-theory (O, Γ), where O is defined in L, LF, AF, G, S, F, GF, SF, WS, or WA, a CQ Q, and a tuple $\mathbf{a}$ over $\Delta \cup \Delta_N$, deciding whether $\mathbf{a}$ is in the skyline answer for q to (O, Γ) is possible in polynomial time in the data complexity.

The above data-tractability results do not carry over to WG, where BCQ^N answering in O is data-complete for EXP, and data-hardness holds even for ground atomic BCQs: so, data-completeness for EXP can be proved similarly to Theorem 5.3.7.

5.3.3 Special cases of dominance testing

As we saw in Section 2.2.2.1, there are many cases when one can obtain polynomial-time complexity results for testing dominance of two outcomes. For example, one

can also use the tractable subclass of CP-theories or approximation algorithms with polynomial-time dominance testing.

Theorem 5.3.11. Let $\mathcal{T}$ be a class of OCP-theories (O, Γ) where BCQ^N answering in O is complete for a deterministic complexity class $\mathcal{C}$ in P, dominance testing can be computed in P, and hardness holds even for ground atomic BCQs. Then, deciding whether

(a) some given $(O, \Gamma) \in \mathcal{T}$ is consistent, and
(b) o dominates o' in (O, Γ), given $(O, \Gamma) \in \mathcal{T}$ and two outcomes o, o',

are both complete for $\mathcal{C}$.

Proof.

(a) **Membership.** By using similar arguments as in the proof for membership of Theorem 5.3.2, one requires only to decide BCQ^{N}s to O (for deciding if an outcome is consistent, and whether two consistent outcomes are equivalent), which is in $\mathcal{C}$, and deciding dominance between two outcomes in our case, which is in P. Overall, this is possible in $\mathcal{C}$ since $\mathcal{C} \supseteq$ P.

Hardness. Hardness for $\mathcal{C}$ follows the same arguments as in the proof of Theorem 5.3.2. We use the same reduction of the consistency problem from that of answering ground atomic BCQs for various languages.

(b) **Membership.** As for membership, by Theorem 5.1.2, it is sufficient to decide whether o and o' are consistent and whether $[o]_\sim \succ [o']_\sim$. As argued in (a), this is overall possible in $\mathcal{C}$.

Hardness. Hardness for $\mathcal{C}$ follows the same arguments as in the proof of Theorem 5.3.2 by using same reduction from the problem of answering ground atomic BCQs.

□

We next prove the results of deciding membership to skyline answers for CQs to consistent OCP-theories when the BCQ^N answering is complete for $\mathcal{C}$ and dominance testing can be computed in P.

Theorem 5.3.12. Let $\mathcal{T}$ be a class of consistent OCP-theories (O, Γ), where BCQ^N answering in O is complete for a deterministic complexity class $\mathcal{C}$ in P, and dominance testing of two outcomes can be done in P. Then, given $(O, \Gamma) \in \mathcal{T}$, a CQ Q, and a tuple $\mathbf{a}$ over $\Delta \cup \Delta_N$, deciding whether $\mathbf{a}$ is in the skyline answer for q to (O, Γ) is complete for $\mathcal{C}$.

Proof. **Membership.** We can use the same arguments as in the membership proof of Theorem 5.3.5, where we use a bounded number calls to BCQ answering for O (which is in $\mathcal{C}$) and dominance testing (which is in P). Therefore, the overall problem of deciding membership to the skyline lies in $\mathcal{C}$.

Hardness. Hardness for $\mathcal{C}$ follows from the more specialised problem of BCQ^N answering being $\mathcal{C}$-hard. We can use the same reduction from the problem of answering ground atomic BCQs as in the proof of Theorem 5.3.5. □

5.4 Conclusion

This chapter introduced ontological CP-theories (OCP-theories) and their fragment OCP-nets, which are a novel combination of Datalog$^{\pm}$ ontologies with CP-theories. I have defined skyline and k-rank answers for CQs to OCP-theories, and have also provided an algorithm for computing such skyline and k-rank answers. Furthermore, I have provided a host of precise complexity (including several data-tractability) results for deciding consistency, dominance, and CQ skyline membership for OCP-theories.

OCP-theories are a more compact language than PP-Datalog$^{\pm}$ with good complexity results for some of their fragments. Moreover, its expressibility allow us to express context-based preferences (e.g., "If the property is in Oxford, then I prefer to rent a room in a college, rather than renting a room in a hotel.").

Interesting topics for future research are to extend the language to express preferences of a group of users, to add uncertainty, and the ability to express negative preferences as well—that is, what the users do not want in their top answers.

Part IV

Score Based Preferences for the Semantic Web

Chapter 6

Single User Quantitative Preferences for the Semantic Web

Chapter 3 studied the complexity of the single-user case of qualitative preferences, showing intractability for CQs, and motivating the search for tractable special cases; then, in Chapter 4, I developed algorithms to answer k-rank queries for unions of atomic queries under group preferences and uncertainty in Datalog$^{\pm}$ [31] ontologies, where the preferences of every user are expressed as strict partial orders (irreflexive and transitive binary relations). The algorithms in Chapter 4 are not optimised for score-based preferences, as they do not leverage this simpler structure. As it is shown below, the special case of score-based preferences allows us to compute answers in polynomial time.

I propose an approach to top-k query answering under user preferences in Datalog$^{\pm}$ ontologies, where the queries are unions of conjunctive queries with safe negation, and the preferences are defined via numerical scores. I develop an algorithm for top-k query answering in this framework, which is based on a generalisation of the previous RankJoin algorithm [85] to ontology-based data access and to UCQs with safe negation as queries.

The rest of this chapter is organised as follows. Section 6.1 introduces SP-

Datalog$^{\pm}$, its syntax and the semantics. Section 6.2 presents algorithms for query answering, along with correctness and data tractability results. Finally, Section 6.3 summarises the main results of this chapter and gives an outlook on future research.

6.1 Formalism

This section adapts the extension of Pref-Datalog$^{\pm}$ in [115] and presented in Section 2.2.3 to the special case where the preference model is *score-based*, i.e., it is an assignment of a numeric score to each element in $\mathcal{H}$ in such a way that $a_1 \succ a_2$ if $score(a_1) > score(a_2)$ (also called *strict weak orderings*). In the sequel, I only refer to such score functions, and the corresponding preference relation is implicit.

Definition 6.1.1. A *score preference-based Datalog*$^{\pm}$ (SP_Datalog$^{\pm}$) ontology (or knowledge base) $KB = (O, score)$ consists of a Datalog$^{\pm}$ ontology O, and a score function *score* from its extended Herbrand base $\mathcal{H}$ to $[0, 1]$.

Below we see an example of such an *SP_Datalog*$^{\pm}$ ontology.

Example 6.1.1. Consider the Datalog$^{\pm}$ ontology from Example 2.1.1. Suppose we have a score function for atoms from the extended Herbrand base $\mathcal{H}$ over the predicates *property*, *user*, *propertyFeature*, *offer*, and *friend*; Table 6.1 shows sample scores of three different such functions. ∎

6.2 Algorithms and Complexity

This section explores how to obtain k-rank answers to unions of conjunctive queries with safe negation (UNCQs).

As a first step, the query-relevant part of the Datalog$^{\pm}$ ontology is *materialised*, i.e., given an ontology $KB = ((D, \Sigma), score)$, we obtain one of the form $KB' = ((D', \emptyset), score)$, on which the query can be equivalently evaluated. For guarded Datalog$^{\pm}$ ontologies, this is possible in polynomial time (and thus the materialisation has also a polynomial size) in the data complexity (where the schema $\mathcal{R}$, the set

Table 6.1: Assignment of scores to atoms from Example 6.1.1

	property				Score functions		
	id	*type*	*location*	*class*	$score_{u_1}$	$score_{u_2}$	$score_{u_3}$
t_1	p_1	apt	l_1	e	0.8	0.3	0.7
t_2	p_2	house	l_2	l	0.7	0.6	0.65
t_3	p_3	apt	l_1	e	0.6	0.2	0.1
t_4	p_4	apt	l_2	l	0.2	0.9	0.7
t_5	p_5	apt	l_2	e	0.2	0.2	0.5
t_6	p_6	apt	l_3	e	0.0	0.0	0.0

	user		Score functions		
	id	*review*	$score_{u_1}$	$score_{u_2}$	$score_{u_3}$
t_7	b	p	0.8	0.5	0.75
t_8	j	n	0.5	0.2	0.4
t_9	a	p	0.1	0.1	0.2

	propertyFeature		Score functions		
	property	*feature*	$score_{u_1}$	$score_{u_2}$	$score_{u_3}$
t_{10}	p_1	g	0.8	0.2	0.4
t_{11}	p_2	p	0.6	0.5	0.75
t_{12}	p_3	p	0.6	0.2	0.4
t_{13}	p_3	g	0.6	0.2	0.4

	offer		Score functions		
	user	*property*	$score_{u_1}$	$score_{u_2}$	$score_{u_3}$
t_{14}	b	p_2	0.7	0.4	0.3
t_{15}	b	p_1	0.6	0.2	0.3
t_{16}	j	p_4	0.6	0.9	0.6
t_{17}	j	p_5	0.5	0.1	0.65

	friend		Score functions		
	user	*user*	$score_{u_1}$	$score_{u_2}$	$score_{u_3}$
t_{18}	b	a	0.7	0.7	0.7
t_{19}	j	a	0.7	0.7	0.7
t_{20}	b	m	0.8	0.6	0.1

Σ of TGDs and NCs, and the query size are all fixed) by computing the guarded chase forest up to a certain (query-dependant) depth. *This materialised database is equivalent to the original ontology in that it can be used to evaluate all CQs that are of bounded width, fixed, or atomic, and thus no loss of expressive power is suffered by taking this step.* This evaluation can clearly be done in polynomial time in the data complexity; however, although the materialised database has a polynomial size, it may be quite large in general, which motivates realising the score-based evaluation of unions of CQs with safe negation via a combination with a corresponding generalisation of the RankJoin algorithm [85], to make the evaluation more efficient. In the rest of this chapter, whenever I refer to a Datalog$^{\pm}$ ontology O or a SP_Datalog$^{\pm}$ ontology $KB = (O, score)$, we will assume that O is already materialised unless stated differently.

6.2.1 Union of Conjunction Queries

I focus on unions (disjunctions) of conjunctive queries that admit negated atoms. As usual, I assume that the negation is safe, which intuitively means that the domain of arguments of a negated atom is restricted to the domain of arguments of the positive atoms.

Definition 6.2.1. A *CQ with safe negation (NCQ)* has the form $q(\mathbf{X}) = \exists \mathbf{Y} R_1(\mathbf{Z}_1) \wedge \cdots \wedge R_m(\mathbf{Z}_m) \wedge \neg N_{m+1}(\mathbf{Z}_{m+1}) \wedge \cdots \wedge \neg N_{m+n}(\mathbf{Z}_{m+n})$, where (i) R_i and N_j are predicates from $\mathcal{R}$, (ii) $\mathbf{Z}_j$ are tuples of variables over $\mathbf{X} \cup \mathbf{Y}$ such that $\mathbf{X} \subseteq \mathbf{Z}_1 \cup \cdots \cup \mathbf{Z}_m$, and (iii) for every negated atom N_i, it holds that $\mathbf{Z}_i \subseteq \mathbf{Z}_1 \cup \cdots \cup \mathbf{Z}_m$. The *positive part* of $q(\mathbf{X})$, denoted $Pos(q(\mathbf{X}))$, is defined as $\exists \mathbf{Y} R_1(\mathbf{Z}_1) \wedge \cdots \wedge R_m(\mathbf{Z}_m)$. A *union of CQs with safe negation (UNCQ)* has the form $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, where each $q_i(\mathbf{X})$ is an NCQ. Its *positive part*, denoted $Pos(q(\mathbf{X}))$, is defined as $\bigvee_{i=1}^{l} Pos(q_i(\mathbf{X}))$.

Before defining the answers to NCQs and UNCQs, I introduce the notion of joined tuple and their scores.

Definition 6.2.2. Given an SP_Datalog$^{\pm}$ ontology $KB = (O, score)$ and an NCQ $q(\mathbf{X}) = \exists \mathbf{Y} R_1(\mathbf{Z}_1) \wedge \cdots \wedge R_m(\mathbf{Z}_m) \wedge \neg N_{m+1}(\mathbf{Z}_{m+1}) \wedge \cdots \wedge \neg N_{m+n}(\mathbf{Z}_{m+n})$, a *joined tuple* $\hat{t}$ of q over KB is a sequence of tuples $t_1, \ldots, t_m$ over $\Delta \cup \Delta_N$ where a homomorphism $h\colon \mathbf{Z}_1 \cup \cdots \cup \mathbf{Z}_m \rightarrow \Delta \cup \Delta_N$ exists such that, for every $i \in \{1, \ldots, m\}$, $h(\mathbf{Z}_i) = t_i$ for some $R_i(t_i) \in O$. Each such $\hat{t}$ is associated with the *score* $\hat{s} = min(min_{i=1}^{m} score(t_i), min_{j=1}^{n} score'(t_{m+j}))$, where $t_{m+j} = h(\mathbf{Z}_{m+j})$ and $score'(t_{m+j}) = 1 - score(t_{m+j})$, if $N_{m+j}(t_{m+j}) \in O$, and $score'(t_{m+j}) = 1$, otherwise, for all $j \in \{1, \ldots, n\}$.

Example 6.2.1. Consider the following NCQ $q_1(U, P) = \exists T, L, C, R, property(P, T, L, C) \wedge \neg offer(U, P) \wedge user(U, R))$ over the relations in Table 6.1, which asks for the pairs of users U and properties P, such that U does not offer property P for rent, and consider the score function $score_{u_1}$. Then, $\hat{t} = t_1, t_7$ is a joined tuple of q_1; it comes from joining $t_1 = (p_1, apt, l_1, e)$ from relation *property* and $t_7 = (b, p)$ from *user*. As there is tuple $t_{15} = (b, p_1)$ in *offer* that matches the values for variables U and P set by $\hat{t}$, its score is $min(0.8, 1 - 0.6, 0.8) = 0.4$. ∎

Let us denote with $J(q, KB)$ the set of pairs $\langle \hat{t}, \hat{s} \rangle$ such that $\hat{t}$ is a joined tuple of q over KB, and $\hat{s}$ is its score, and by $J_t(q, KB)$ the set of pairs $\langle \hat{t}, \hat{s} \rangle$ of $J(q, KB)$ such that the projection of $\hat{t}$ yields t. We are now ready to define the answers to NCQs.

Definition 6.2.3. Given an SP_Datalog$^{\pm}$ ontology KB and an NCQ $q(\mathbf{X})$, the set of *answers* to q over KB, denoted $ans(q, KB)$, is the set of all $\langle t, p \rangle$ such that (i) t is a tuple over Δ, and the projection of some joined tuple $\hat{t}$ of q over KB onto $\mathbf{X}$, and (ii) $p = max_{\langle \hat{t}, \hat{s} \rangle \in J_t(q(\mathbf{X}), KB)} \hat{s}$.

According to Definitions 6.2.2 and 6.2.3, my preference-based framework allows a tuple t to be an answer to an NCQ even if t satisfies the negated atoms. Nevertheless, its score is computed in a way that, if the scores $score(t_{m+j})$ (i.e., the scores of the tuples of the negated relations N_{m+j} whose join and projection yields t) in KB are high, then it is less likely that t appears in the top-k answers, as $1 - score(t_{m+j})$ is considered in the computation of the score of t instead of $score(t_{m+j})$.

Intuitively, an answer to an UNCQ q is a pair $\langle t, s\rangle$ such that t is an answer to some NCQ q_i in q, and s is the maximum score of t among all q_i's.

Definition 6.2.4. Given an SP_Datalog$^{\pm}$ KB and a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, the set of *answers* to q over KB, denoted $ans(q, KB)$, is the set of all $\langle t, g\rangle$ such that there exists $\langle t, p\rangle \in \bigcup_{i=1}^{l} ans(q_i, KB)$ with $g = max\{p \mid \langle t, p\rangle \in \bigcup_{i=1}^{l} ans(q_i, KB)\}$.

Let us next define top-k answers to UNCQs, which are intuitively at most k answers whose scores are always above or equal to the score of all non-top-k answers.

Definition 6.2.5. Given an SP_Datalog$^{\pm}$ ontology KB and a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, a *top-k answer* to q over KB, denoted $ans_k(q, KB)$, is a sequence of pairs $\langle t, s\rangle$ of atoms and scores such that:

(i) $ans_k(q, KB) \subseteq ans(q, KB)$,

(ii) $|ans_k(q, KB)| = min(k, |ans(q, KB)|)$, and

(iii) $s \geq s'$ for all $\langle t, s\rangle \in ans_k(q, KB)$ and $\langle t', s'\rangle \in ans(q, KB) \setminus ans_k(q, KB)$.

In the rest of this chapter, I refer to "a top-k answer", since ties in scores can lead to different sequences satisfying the conditions in the definition; I use notation $pos(a, s)$ to refer to the position of element a in sequence s. For simplicity, we also slightly abuse notation by sometimes referring to answers as sequences of atoms (without the scores), and sometimes treat sequences as sets.

6.2.2 Computing Top-k Answers

My current approach extends the RankJoin operator in [85] to work with UNCQs. The original RankJoin operator takes as input an integer k, a positive conjunctive query, a monotonic ranking function f, and m relations (where each tuple in each relation is accompanied by a score), and it yields the top-k joined tuples over the m relations, in descending order of their combined scores (computed using f). Specifically, it assumes that each of the m input relations is sorted by tuple-score in

descending order, and tuples from the m relations are scanned, and joined until k joined tuples are found such that the lowest among their scores is greater than or equal to a certain threshold.

The ranking function f computes the scores $\hat{s}$, p, and g of tuples by applying the *min* and *max* operators to the scores of the atoms of $\mathcal{H}$, as defined in Definitions 6.2.2, 6.2.3, and 6.2.4, respectively. This generalises conjunctions and disjunctions in classical logic; the framework can be easily adapted to other score computations.

Algorithm 11 Algorithm for computing a k-rank answer to UNCQ q .

Input: (i) SP_Datalog$^{\pm}$ ontology $KB = (O, score)$,
(ii) UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$;
(iii) integer $k > 0$.
Output: a top-k answer to $q(\mathbf{X})$.
Called by: Algorithms 19 and 20.
Calls: Algorithms 12 to 16.
Global Variables: $relList$, $hashTab$, Its, $topK$.

```
1: function RankJoinUNCQ(KB, q, k)
2:     querySet ← {q_1(X), ..., q_l(X)};
3:     topK ← ∅;
4:     InitStructures(querySet); //Algorithm 12
5:     while CanContinue(querySet, k) do //Algorithm 13
6:         q ← chooseQuery(querySet);
7:         TS ← getTuples(q); // Algorithm 14
8:         if (TS = ∅) then //all the iterators in Its(q) reached the end
9:             remove q from querySet
10:        else
11:            for each ⟨t, p⟩ in TS do
12:                UpdateTopK(⟨t, p⟩); //Algorithm 15
13:        T ← getThreshold(q); // Algorithm 16
14:        if (|topK| = k and T ≤ minScore(topK)) then
15:            remove q from querySet if present;
       return topK.
```

6.2.2.1 Algorithm for Computing Top-k Answers

Algorithm 11 presents the way one computes the top-k answers for a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$ over $KB = (O, score)$. The function yields a set $topK$ of pairs $\langle t, g\rangle$ representing a top-k answer to $q(\mathbf{X})$ over KB, according to Definition 6.2.5. I denote with $R_1, \ldots, R_m$ the relations in O over which $Pos(q_i(\mathbf{X}))$ is formulated.

Variables and initialisation. RankJoinUNCQ uses the two local variables *querySet* and *topK*, where *querySet* stores the NCQs $q_i(\mathbf{X})$ composing the input query $Q(\mathbf{X})$, and *topK* stores the set of pairs $\langle t, g\rangle$ that represent the top-k answers to $Q(\mathbf{X})$ computed so far. Furthermore, I assume three global variables *relList*, *hashTab*, and *Its*, which are mappings from elements in *querySet* into lists of relation names, sets of hash tables, and sets of iterators, respectively. InitStructures, called in line 3 of RankJoinUNCQ (Algorithm 11), initialises these structures as follows (see Algorithm 12): for each NCQ q_i in *querySet*, *relList* contains the list with the (names of the) m relations $R_1, \ldots, R_m$ that appear in $Pos(q_i)$ (these are the positive relations over which q_i is evaluated), *hashTab* contains a hash table for each R_j in *relList*, which is used to store the tuples of R_j that have already been scanned, and *Its* contains a set of iterators, one for each table R_j in $relList(q_i)$; iterator It_j in $Its(q_i)$ points to the next tuple to be scanned in R_j in the evaluation of q_i, so it is initialised to the first tuple in R_j. Note that if q_i contains the same relation R multiple times (self-joins), *relList* accordingly contains R multiple times, and $Its(q_i)$ contains an iterator for each occurrence of R. Note also that the algorithm deals with special types of iterators that in the call to getNext always retrieve the maximum tuple from the ones not parsed yet. One example of how this can be done is to sort the relations, and so the retrieval of the next tuple by the iterators can be done in constant time.

Main loop Algorithm 11. At each iteration, RankJoinUNCQ randomly chooses an NCQ q from *querySet* via chooseQuery (line 5), and retrieves a set TS of pairs $\langle t, p\rangle$ by calling getTuples (Algorithm 14, line 6), which deals with the progressive evaluation of the NCQ q. If TS is empty (line 7), i.e., no more tuples can be obtained for the evaluation of q, RankJoinUNCQ removes q from *querySet* (line 8). Otherwise, UpdateTopK (Algorithm 15) is called for each pair $\langle t, p\rangle$ to update the current top-k answers, if necessary. Note that each relation is sorted by score from maximum to minimum; thus, at each iteration, getTuples returns the answers to q with the

Algorithm 12 Initialisation procedures.

Input: $querySet = \{q_1(\mathbf{X}), \ldots, q_l(\mathbf{X})\}$.
Called by: Algorithm 11.
Global Variables: $relList$, $hashTab$, Its, $topK$.

```
1:  procedure InitStructures(querySet).
2:      relList ← ∅; //empty mapping from queries to lists of relations
3:      hashTab ← ∅; //empty mapping from queries to sets of hash tables
4:      Its ← ∅; //empty mapping from queries to sets of iterators
5:      for all q_i ∈ querySet do
6:          for all relation R_j appearing in Pos(q_i(X)) do
7:              relList(q_i).append(R_j); //add R_j to relList(q_i)
8:              H_j ← ∅; //create an empty hash table
9:              hashTab(q_i).append(H_j); //add empty hash table H_j to hashTab(q_i)
10:             new.It_j; //create It_j and set it to the first tuple in R_j
11:             Its(q_i).append(It_j); //add It_j to Its(q_i)
```

highest possible score. Thus, if $topK$ contains already k answers, and getThreshold(q) (Algorithm 16) is lower than the minimum score of the answers in $topK$, then none of the rest of the answers to q can contribute to the top-k answers, and therefore q is removed from $querySet$ (line 14). The value getThreshold(q), computed as shown in Algorithm 16, is the maximum tuple score obtainable in subsequent steps of the evaluation of q. Let T be the maximum tuple score obtainable in subsequent steps of the evaluation of the *whole* query $Q(\mathbf{X})$, computed by taking the maximum of the values provided by getThreshold(q) for every NCQ q in $querySet$. Algorithm RankJoinUNCQ halts when (a) the lowest score of $topK$ is greater than T, and k answers have been found, or when (b) $querySet$ becomes empty, i.e., no query is left to be evaluated; the tests for these conditions (while-loop condition in line 4) are implemented in canContinue in Algorithm 13. In every case, RankJoinUNCQ returns the set $topK$ (line 15).

Function getTuples. The pseudocode for this subroutine is shown in Algorithm 14; it deals with the progressive evaluation of an NCQ q, i.e., it progressively computes $ans(q, KB)$ as formalised in Definition 6.2.3. At each invocation, it yields a set TS of pairs $\langle t, p\rangle$ such that (i) t is an instance of the free variables in $\mathbf{X}$ obtained

from $R_1, \ldots, R_m$ following the RankJoin strategy, and (ii) p is the computed score for t. As the first step, getTuples retrieves $H_1, \ldots, H_m$, and $It_1, \ldots, It_m$ with the corresponding values for q from the global variables. The while-loop in line 5 tries to find a next relation to scan: if $relList(q)$ is not empty, then a relation $currRel$ is randomly chosen from it; if this relation has already been fully scanned (line 8), then it is removed from $relList(q)$, and the process is repeated until a relation that still has tuples to scan is found. If, however, all the relations have been entirely scanned, then all the possible join combinations have been tried, and no more tuples can be provided for the evaluation of $q(\mathbf{X})$, thus getTuples returns the empty set (line 10). Otherwise, $currRel$ holds the next relation R_i to scan; function getNext(It_i) fetches the pair $\langle t, score(t)\rangle$ where t is the next tuple in R_i that has not been scanned yet, and $score(t)$ is the corresponding score for t in R_i; note that as tuples in R_i are sorted in descending order relative to their scores, t has a score higher or equal than the scores of the tuples that were not chosen yet by the iterator in R_i. The pair $\langle t, score(t)\rangle$ provided by getNext(It_i) is added to H_i (line 13)—recall that H_i is the hash table associated with R_i, and join is invoked afterwards (line 14).

Function join. This function builds a set J of joined tuples resulting from joining tuple t with the tuples already in $H_1, \ldots, H_{i-1}, H_{i+1}, \ldots H_m$, according to q; join effectively computes the join in q for t with all the tuples from the other relations that have already been scanned.

Function computeScore. This function (Algorithm 18), invoked on a set of joined tuples $\hat{t}$, and an NCQ $q(\mathbf{X})$, computes $\hat{s}$ as in Definition 6.2.2 for each $\hat{t}$ (line 15).

Procedure UpdateTS. For each of the obtained joined tuples $\hat{t}$, procedure UpdateTS is called (line 17 of getTuples). UpdateTS in Algorithm 17 projects $\hat{t}$ on the attributes provided by $Att(\mathbf{X})$, which are the attributes of the relations of $q(\mathbf{X})$ over which the variables in $\mathbf{X}$ are mapped to, yielding t. Next, UpdateTS checks whether t occurs in TS with a score p lower than $\hat{s}$: in this case, the score p of t in TS is updated with $\hat{s}$. In the case that t does not appear in TS, the pair $\langle t, \hat{s}\rangle$ is added to TS. In

line 18, function getTuples returns set TS as output.

Algorithm 13 Stopping the loop in the main algorithm.

Input: (i) a set of NCQs $querySet$, and
(ii)$k > 0$.
Output: *true* if it can continue, $false$ otherwise.
Called by: Algorithm 11.
Calls: Algorithm 16.
Global Variables: $relList$, $hashTab$, Its, $topK$.

```
1: function canContinue(querySet, k)
2:     if querySet = ∅ then return false;
3:     T ← max{getThreshold(q_i) | q_i ∈ querySet}; //Algorithm 16
4:     if (|topK| = k and minScore(topK) ≥ T) then return false;
       return true.
```

Procedure UpdateTopK**.** The procedure (Algorithm 15) first checks if t is already in $topK$, and, if so, then it updates the score of t in $topK$, if necessary (line 3)—it keeps the maximum score for t according to Definition 6.2.5. Otherwise, if $topK$ does not have k pairs yet, then $\langle t, p\rangle$ is added to $topK$, else if $|topK| = k$, and p is greater than the minimum score of the tuples in $topK$, then $\langle t', g'\rangle$ in $topK$ such that $minScore(topK) = g'$ is replaced with $\langle t, p\rangle$.

	$querySet$	getTuples	$top-1$	$T =$ getThreshold(q_1)
1	$\{q_1\}$	$\emptyset$	$\emptyset$	$max\{min\{0.8, 0.8, 1\}, min\{0.8, 0.8, 1\}\} = 0.8$
2	$\{q_1\}$	$\{\langle(b, p_1), 0.4\rangle\}$	$\{\langle(b, p_1), 0.4\rangle\}$	$max\{min\{0.8, 0.8, 1\}, min\{0.8, 0.8, 1\}\} = 0.8$
3	$\{q_1\}$	$\{\langle(b, f_2), 0.3\rangle\}$	$\{\langle(b, p_1), 0.4\rangle\}$	$max\{min\{0.7, 0.8, 1\}, min\{0.8, 0.8, 1\}\} = 0.7$
4	$\{q_1\}$	$\{\langle(j, p_1), 0.5\rangle, \langle(j, p_2), 0.5\rangle\}$	$\{\langle(j, p_1), 0.5\rangle\}$	$max\{min\{0.7, 0.8, 1\}, min\{0.5, 0.8, 1\}\} = 0.7$
5	$\{q_1\}$	$\{\langle(b, p_3), 0.6\rangle, \langle(j, p_3), 0.5\rangle\}$	$\{\langle(b, p_3), 0.6\rangle\}$	$max\{min\{0.6, 0.8, 1\}, min\{0.5, 0.8, 1\}\} = 0.6$

Figure 6.1: Trace of RankJoinUNCQ for $q_1(U, P)$ for user u_1 (Example 6.2.2).

The following example shows how RankJoinUNCQ works.

Algorithm 14 Function evaluating a NCQ.

Input: NCQ $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$.
Output: Set TS of pairs $\langle t, p \rangle$, where t is an instance of the free variables in $\mathbf{X}$, and p is its score.
Called by: Algorithm 11.
Calls: Algorithms 17 and 18.
Global Variables: *relList*, *hashTab*, *Its*, *topK*.

```
1:  function getTuples(q)
2:      {H_1, ..., H_m} ← hashTab(q);  //hash tables over R_1, ..., R_m ∈ relList(q)
3:      {It_1, ..., It_m} ← Its(q);  //iterators over rel. R_1, ..., R_m in relList(q)
4:      TS ← ∅;
5:      currRel ← ∅;
6:      done ← false;
7:      while relList(q) ≠ ∅ and not done do
8:          currRel ← chooseRel(relList(q));
9:          Let i be such that currRel = R_i;
10:         if not hasNext(It_i) then
11:             remove R_i from relList(q); //R_i was scanned
12:         else done ← true;
13:     if relList(q) = ∅ then return ∅;
14:     Let i be such that currRel = R_i;
15:     ⟨t, score(t)⟩ ← getNext(It_i); //next element of the It_i iterator
16:     put ⟨t, score(t)⟩ in H_i;
17:     J ← join(Φ, H_1, ..., H_{i-1}, t, H_{i+1}, ..., H_m);
18:     JP ← computeScore(J, q);  //Algorithm 18
19:     for each ⟨t̂, ŝ⟩ ∈ JP do
20:         UpdateTS(⟨t̂, ŝ⟩, TS);  //Algorithm 17
        return TS.
```

.

Algorithm 15 Updating *topK* with a new scored tuple.

Input: $\langle t, p\rangle$ a scored tuple.
Called by: Algorithm 11.
Global Variables: *relList*, *hashTab*, *Its*, *topK*.

```
1: procedure UpdateTopK(⟨t, p⟩)
2:     if there exists (⟨t, g⟩ ∈ topK) then
3:         if p ≥ g then
4:             replace ⟨t, g⟩ in topK with ⟨t, p⟩;
5:         else
6:             if |topK| < k then
7:                 add ⟨t, p⟩ to topK;
8:             else
9:                 minimum = minScore(topK);
10:                if minimum < p then
11:                    remove a tuple from topK with score minimum;
12:                    add ⟨t, p⟩ to topK.
```

Algorithm 16 Computing the threshold of the score of the unseen tuples.

Input: NC $q(\mathbf{X}) = \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$ where $Pos(q(\mathbf{X}))$ contains m atomic relations $R_1, \ldots, R_m$ ($R_i \in \mathcal{R}$).
Output: $T \geq 0$.
Called by: Algorithms 11 and 13.
Global Variables: *relList*, *hashTab*, *Its*, *topK*.

```
1: function getThreshold(q)
2:     for all i ∈ {1, 2 ... m} do
3:         Let r_i^1 be the first tuple of R_i;
4:         Let r_i^last be the last scanned tuple of R_i;
5:         T_i ← min{score(r_1^1), ..., score(r_{i-1}^1), score(r_i^last), score(r_{i+1}^1), ...,
6:             score(r_m^1), 1};
7:     T ← max{T_1, ..., T_m}.
8: return T.
```

Algorithm 17 Updating TS with a new scored tuple.

Input: (i) A pair $\langle \widehat{t}, \hat{s}\rangle$, and
(ii) a set of score tuples TS that keeps modifications outside this procedure.
Called by: Algorithm 14.
Global Variables: *relList*, *hashTab*, *Its*, *topK*.

```
1: procedure UpdateTS(⟨t̂, ŝ⟩)
2:     t ← project(t̂, Att(X));
3:     if ⟨t, p⟩ ∈ TS and ŝ ≥ p then
4:         update p with ŝ;
5:     if ⟨t, p⟩ ∉ TS then
6:         put ⟨t, ŝ⟩ in TS.
```

Algorithm 18 Computing the scores of joined tuples.

Input: (i) A set J of $\widehat{t}$ where $\widehat{t} = t_1, \ldots, t_m \mid \langle t_j, score(t_j)\rangle \in H_j$, and (ii) a UNCQ $q(\mathbf{X}) = \exists \mathbf{Y} R_1(\mathbf{Z}_1) \wedge \cdots \wedge R_m(\mathbf{Z}_m) \wedge \neg N_{m+1}(\mathbf{Z}_{m+1}) \wedge \cdots \wedge \neg N_{m+n}(\mathbf{Z}_{m+n})$.
Output: a set JP of pairs $\langle \hat{t}, \hat{s}\rangle$.
Called by: Algorithm 14.
Global Variables: $relList$, $hashTab$, Its, $topK$.

```
1:  function computeScore(J, q)
2:      JP ← ∅;
3:      for all t̂ ∈ J do
4:          ŝ ← min(score(t_j), ..., score(t_m));
5:          for each N_j do
6:              q_j(X) ← ∃Y R_1(Z_1) ∧ ··· ∧ R_m(Z_m) ∧ N_j(Z_j);
7:              if (there exists tuple t_j in N_j such that the projection of t_1, ..., t_m,
8:                t_j onto X is an answer to q_j) then
9:                  ŝ ← min{ŝ, 1 − score(t_j)}
10:             else ŝ ← min{ŝ, 1};
11:         add ⟨t̂, ŝ⟩ to JP;
        return JP.
```

	R_i	It_{user}, $It_{property}$, H_{user}, $H_{property}$	$\hat{t}$, with $\langle \hat{t}, \hat{s}\rangle \in J$	$\hat{s}$, with $\langle \hat{t}, \hat{s}\rangle \in J$	TS
1	*user*	t_8, t_1, $\{\langle t_7, 0.8\rangle\}$, $\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
2	*property*	t_8, t_2, $\{\langle t_7, 0.8\rangle\}$, $\{\langle t_1, 0.8\rangle\}$	$\{\hat{t} = t_7, t_1\}$	$\{min\{0.8, 1-0.6, 0.8\} = 0.4\}$	$\{\langle (b, p_1), 0.4\rangle\}$ $\{\langle (b, p_1), 0.4\rangle\}$
3	*property*	t_8, t_3, $\{\langle t_7, 0.8\rangle\}$, $\{\langle t_1, 0.8\rangle, \langle t_2, 0.7\rangle\}$	$\{\hat{t} = t_7, t_2\}$	$\{min\{0.8, 1-0.7, 0.7\} = 0.3\}$	$\{\langle (b, p_2), 0.3\rangle\}$ $\{\langle (b, p_2), 0.3\rangle\}$
4	*user*	t_9, t_3, $\{\langle t_7, 0.8\rangle, \langle t_8, 0.5\rangle\}$, $\{\langle t_1, 0.8\rangle, \langle t_2, 0.7\rangle\}$ $\{\langle t_1, 0.8\rangle, \langle t_2, 0.7\rangle\}$	$\{\hat{t_1} = t_8, t_1, \hat{t_2} = t_8, t_2\}$ $\hat{t_2} = t_8, t_2\}$	$\{min\{0.5, 1, 0.8\} = 0.5, min\{0.5, 1, 0.7\} = 0.5\}$ $min\{0.5, 1, 0.7\} = 0.5\}$	$\{\langle (j, p_1), 0.5\rangle, \langle (j, p_2), 0.5\rangle\}$ $\langle (j, p_2), 0.5\rangle\}$
5	*property*	t_9, t_4, $\{\langle t_7, 0.8\rangle, \langle t_8, 0.5\rangle\}$ $\{\langle t_1, 0.8\rangle, \langle t_2, 0.7\rangle, \langle t_3, 0.6\rangle\}$	$\{\hat{t_1} = t_7, t_3, \hat{t_2} = t_8, t_3\}$	$\{min\{0.6, 1, 0.8\} = 0.6, min\{0.6, 1, 0.5\} = 0.5\}$ $min\{0.6, 1, 0.5\} = 0.5\}$	$\{\langle (b, p_3), 0.6\rangle, \langle (j, p_3), 0.5\}$

Figure 6.2: Trace of getTuples on each call for q_1 for user u_1.

6.2.2.2 Running Example of Algorithm 11

Example 6.2.2. Consider again the NCQ $q_1(U, P) = \exists T, L, C, R\,(property(P, T, L, C) \wedge \neg offer(U, P) \wedge user(U, R))$ over the relations in Table 6.1, which asks for the pairs of users U and properties P, such that U does not offer property P for rent, and consider the score function $score_{u_1}$ and assume we are interested in a top-1 answer.

Since $querySet$ contains only q_1, chooseQuery returns always q_1. InitStructures initializes $relList$ (mapping q_1 to $[user, property]$), $hashTab$ (mapping q_1 to the set of

hash tables $\{H_{user}, H_{property}\}$, where both hash tables are empty), and *Its* (mapping q_1 to the set $\{It_{user}, It_{property}\}$, where iterators It_{user} and $It_{property}$ point to tuples t_7 and t_1, respectively).

Figure 6.1 shows each iteration of RankJoinUNCQ, while Figure 6.2 shows each corresponding call of getTuples within the iteration of RankJoinUNCQ. The values in the rows of both tables show values of their variables after finishing each iteration and each function call, respectively.

In the first iteration, getTuples is called, and returns the empty set, as, assuming *user* is chosen, only t_7 is scanned, and no joins are produced yet. In the second call to getTuples, *property* is chosen, and t_1 is scanned. The invocation of join in this case produces the joined tuple $\hat{t} = t_7, t_1 = (b, p, p_1, apt, l_1, e)$. As $t_{15} = (b, p_1)$ is in *offer*, which matches the values for variables P and U set by $\hat{t}$, the score $\hat{s}$ is computed as $min\{0.8, 1 - 0.6, 0.8\} = 0.4$. UpdateTS projects $\hat{t}$ onto $Att(X)$ yielding $t = \langle(b, p_1)\rangle$, which is added to TS with score 0.4 and returned. Then, RankJoinUNCQ adds $\langle\langle(b, p_1)\rangle, 0.4\rangle$ to $topK$, and getThreshold(q_1) yields $T = 0.8$. Since $lowerScore(topK) = 0.4$ is not greater than 0.8, q_1 is not removed from $querySet$, and the algorithm does not stop.

In the third call to getTuples, *property* is chosen, and t_2 is scanned. Then, *join* produces $\hat{t} = t_7, t_2 = (b, p, p_2, house, l_2, l)$ with projection $t = (b, p_2)$ and score $min\{0.8, 1 - 0.7, 0.7\} = 0.3$. Next, $\{\langle(b, p_2), 0.3\rangle\}$ is returned to RankJoin-UNCQ, but $topK$ does not change, as $|top_k| = 1$, and $0.3 \leq minScore(topK) = 0.4$. The call to getThreshold(q_1) produces 0.7, and the algorithm does not halt.

In the fourth call to getTuples, t_{12} is scanned, and join produces $\hat{t_1} = t_8, t_1$ and $\hat{t_2} = t_{182}, t_2$, whose projections onto $Att(X)$ are (j, p_1) and (j, p_2), respectively. As there is no tuple (j, p_1) or (j, p_2) in *offer*, the scores are $min\{0.5, 1, 0.8\} = 0.5$ and $min\{0.5, 1, 0.7\} = 0.5$, respectively. Both projected tuples and their scores are returned. RankJoinUNCQ first scans $\langle(j, p_1), 0.5\rangle$, and since $minScore(topK) = 0.4$, the pair $\langle(b, p_1), 0.4\rangle$ in $topK$ is replaced with $\langle(j, p_1), 0.5\rangle$. Next, (j, p_2) is scanned,

but since its score is not greater than 0.5, $topK$ is not changed. The value of T is computed as $max\{0.7, 0.5\} = 0.7$, and since $\mathsf{minScore}(topK) = 0.5$, RankJoinUNCQ does not halt.

Finally, in the fifth invocation of getTuples, tuple t_3 from *property* is scanned; in this case, two joined tuples $\hat{t_1} = t_7, t_3$ and $\hat{t_2} = t_8, t_3$ are produced; as neither (b, p_3) nor (j, p_3) is in *offer*, getTuples returns $\{\langle(b, p_3), 0.6\rangle, \langle(j, p_3), 0.5\rangle\}$. First, $\langle(b, p_3), 0.6\rangle$ is considered, and therefore $\langle(j, p_1), 0.5\rangle$ in $topK$ is replaced by $\langle(b, p_3), 0.6\rangle$; since the score of (j, p_3) is not greater than 0.6, $topK$ remains unchanged. Then, $\mathsf{getThreshold}(q_1)$ yields $T = max\{0.6, 0.5\} = 0.6$, and since $|topK| = 1$ and $minScore(topK) = 0.6$, then q_1 is removed from $querySet$, and RankJoinUNCQ returns $\{\langle(b, p_3), 0.6\rangle\}$ as the top answer to the query q_1. ∎

6.2.2.3 Correctness of Algorithm 11

The following theorem states that my algorithm is correct.

Theorem 6.2.1. Given a SP_Datalog$^{\pm}$ ontology KB and a union of conjunction queries $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, RankJoinUNCQ correctly computes a top-k answer to $q(\mathbf{X})$.

Proof. All the relations in $Pos(q)$ are assumed to be sorted by score. The position of *tuple* in the relation R is defined by *position*(*tuple*, R). The proof is by contradiction. Suppose the k-th tuple of the $topK$ list yielded by RankJoinUNCQ is t_k, with score g_k.

Assume that there is a joined tuple $\hat{t}$ of some $q_a(\mathbf{X})$, with $a \in [1..l]$, such that its score $\hat{s}$ has $\hat{s} > g_k$, and $\hat{t}$ has not been processed by $\mathsf{getTuples}(q_a(\mathbf{X}))$, thus (the projection of) $\hat{t}$ does not belong to $topK$. Let $q_a(\mathbf{X})$ be of the form $\exists\mathbf{Y} R_1(\mathbf{Z}_1) \wedge \cdots \wedge R_m(\mathbf{Z}_m) \wedge \neg N_{m+1}(\mathbf{Z}_{m+1}) \wedge \cdots \wedge \neg N_{m+n}(\mathbf{Z}_{m+n})$, and $\hat{t}_1, \ldots, \hat{t}_m$ be the tuples of $R_1, \ldots, R_m$, respectively, whose join yields $\hat{t}$, and $score(\hat{t}_1) \ldots, score(\hat{t}_m)$ the scores of $\hat{t}_1, \ldots, \hat{t}_m$ in $R_1, \ldots, R_m$, respectively. For each $j \in [m+1..m+n]$, let us denote as $score(\hat{t}_j)$ the one's complement of the score of tuple $\hat{t}_j$ of N_j, if the

projection of $\hat{t}_1, \ldots, \hat{t}_m, \hat{t}_j$ onto $\mathbf{X}$ is an answer to q_a ($score(\hat{t}_j) = 1$ otherwise).

As RankJoinUNCQ halted, $g_k \geq T$, where T is the threshold computed at line 2 of Algorithm 13. This means that $g_k \geq T_a$, where $T_a = \mathsf{getThreshold}(q_a) = \mathsf{max}\{T_1, \ldots, T_m\}$. Since, by hypothesis, $\hat{s} > g_k$, it follows that $\hat{s} > T_a$ and thus $\hat{s}$ must be greater than every threshold T_j, with $i \in [1..m]$.

Let $\langle r_i^1, score(r_i^1)\rangle$ and $\langle r_i^{last}, score(r_i^{last})\rangle$ be the first and the last scanned tuple of R_i along with their scores, respectively, with $1 \leq i \leq m$, and consider *any* $T_i \in \{T_1, \ldots, T_m\}$. By definition, T_i is equal to $\mathsf{min}\{score(r_1^1), \ldots, score(r_i^{last}) \ldots, score(r_m^1), 1\}$, and the score $\hat{s}$ is computed as $\min\{ score(\hat{t}_1), \ldots, score(\hat{t}_m), score(\hat{t}_{m+1}) \ldots, score(\hat{t}_{m+n})\}$. Since $score(\hat{t}_i) \leq score(r_i^1)$ for every $i \in [1..m]$, and $score(\hat{t}_j) \leq 1$ for every $j \in [m+1..m+n]$, it is easy to see that, in order for $\hat{s} > T_i$ to hold, it must hold that $score(\hat{t}_i) > score(r_i^{last})$. This in turn implies that $position(\hat{t}_i, R_i) > position(r_i^{last}, R_i)$, since R_i is scanned in descending score order. By applying this reasoning to *every* $T_i \in \{T_1, \ldots, T_m\}$ (as $\hat{s}$ must be greater than *every* threshold T_i), we have that $score(\hat{t}_i) > score(r_i^{last})$ for *every* $i \in [1..m]$, implying that $position(\hat{t}_i, R_i) > position(r_i^{last})$ for *every* $i \in [1..m]$. This means that, since every relation is scanned in descending score order, every $\hat{t}_i$ must have been returned by $\mathsf{getNext}(It_i)$ (see line 12 of Algorithm 14) for every $i \in [1..m]$, then the joined tuple $\hat{t}$ must have been produced by $\mathsf{getTuples}(\mathsf{q_a})$, contradicting the initial assumption. □

6.3 Conclusion

Thus chapter proposes an approach to top-k query answering under user preferences in Datalog$^{\pm}$ ontologies, where the queries are unions of conjunctive queries with safe negation, and the preferences are defined via numerical scores. It showed that the special case of score-based preferences allows to compute answers in polynomial time for fragments of Datalog$^{\pm}$ for which query answering can be done in polynomial time.

One topic of ongoing and future work is to further experimentally evaluate

the similarity of preference aggregations to human judgement in order to select the best suited ones for search, and query answering in the Social Semantic Web. Another interesting topic for future research is to generalise the presented approach to ontologies with uncertainty.

Chapter 7

Group of Users Quantitative Preferences for the Semantic Web

This chapter develops a novel integration of ontology languages with score-based preferences of a group of users where the queries are unions of conjunctive queries with safe negation.

The problem of combining individual preferences to produce a ranking of elements based on an aggregated view of the group becomes a central issue. There are two main challenges in accomplishing this: (i) the fact that disagreement inevitably comes up and must be resolved in some principled manner, and (ii) tractability is even harder to accomplish compared to the single-user case (as it involves the additional aggregation of potentially contradicting preferences of a potentially large collection of users).

This is a generalisation for a group of users of SP-Datalog$^{\pm}$ introduced in Chapter 6. It generalise the top-k query answering under the preferences of a group of users (rather than a single user), which involves the aggregation of (potentially conflicting) user preferences.

First, the chapter focuses on the problem of top-k query answering in a setting where a group of users wishes to obtain an answer that somehow addresses all of

their preferences. Then, it studies two different approaches to such aggregations and their properties. Moreover, it provides experimental results on the performance (in terms of running time), and the quality of my algorithms (using standard measures from information retrieval, and it explore which of the techniques for aggregating user preferences is the best and how much their results differ).

The rest of this chapter is organised as follows. In Section 7.1, I introduce SP-Datalog$^{\pm}$, its syntax and the semantics. Section 7.2 presents algorithms for query answering, along with correctness and data tractability results. Section 7.3 contains the semantic properties for the aggregation techniques and a comparison to my previous group-based approach. In Section 7.4, I present my implementation of my SP-Datalog$^{\pm}$ ontology and the results of the experiments in terms of the quality of the results and time performance. Finally, Section 7.5 summarises the main results of this chapter and gives an outlook on future research.

7.1 Formalism

To this end, I extend the framework presented so far to include a set of users and their respective score functions. Given a set of n users $\mathcal{U} = \{u_1, \dots u_n\}$ and $a \in \mathcal{H}$, I denote with $score_{u_i}(a)$ the score assigned by user u_i to an atom a, and assume that this mapping is defined for all pairs of ground atoms entailed by the ontology and users in the group. Here, it is naturally assumed that one global Datalog$^{\pm}$ ontology is common to all users. A score-based group preference GSP-Datalog$^{\pm}$ (group of users score-based preferences Datalog$^{\pm}$) consists of the Datalog$^{\pm}$ ontology, set of users and their respective score functions.

Example 7.1.1. Consider the Datalog$^{\pm}$ ontology from Example 2.1.1. Suppose we have a score function for atoms from the extended Herbrand base $\mathcal{H}$ over the predicates *property*, *user*, *propertyFeature*, *offer*, and *friend*. In the relations in Table 6.1, the last three columns of each table specify the score function for three different users. For instance, for relation *property*, user u_1 assigns a score of 0.8 to $property(f_1, apt, l_1, e)$,

while users u_2 and u_3 assign it 0.3 and 0.7, respectively. Now, consider the query $q_1(U, P) = \exists T, L, C, R\,(\mathit{property}(P, T, L, C) \wedge \neg \mathit{offer}(U, P) \wedge \mathit{user}(U, R))$—the top-1 answer for this query was computed in Example 6.2.2 for u_1; but we would now like to know the top-1 answer considering also the scores from users u_2 and u_3. ∎

The main challenge in query answering considering the preferences of a group of users is that the user preference models may be in disagreement with each other. The study of preference aggregation strategies to address this problem in ontological query answering was first proposed in [120] where an operator is defined for the aggregation of n individual SPOs, each representing the preferences of an individual. The following definition is an adaptation of aggregation operators where the preference models are represented using score functions.

Definition 7.1.1. Given a set R of tuples and n score functions $\mathit{score}_{u_1}, \ldots, \mathit{score}_{u_n}$ over the tuples in R, a *score aggregation operator* $\biguplus(R, \mathit{score}_{u_1}, \ldots, \mathit{score}_{u_n})$ yields a score function score^* over the tuples in R.

These aggregation operators are quite general, asking only to satisfy the property that the resulting preference relation is also a strict weak order; depending on the application, other properties may be desirable, such as the additional ones studied in [120] for SPOs. For the particular setting of score-based preference functions, the problem of aggregating such functions is similar to that of rank fusion that is discussed as related work. In Section 7.4, I select some specific ranking aggregations from the literature and implement them in my framework.

7.2 Query Answering

In the rest of this section, I analyse two approaches to computing an answer to a top-k query for a set of users. The first one applies an aggregation operator to the result of issuing the query to all users separately to obtain a single top-k answer; I refer to this as the *aggregation-last* approach. The second approach, called

aggregation-first, applies the aggregation operator to the input tables, effectively merging the users' preferences into a single score assignment. The latter approach can be seen as the construction of a single virtual user that aggregates the preferences of all the individuals from the group; the top-k answers are then computed over this new score assignment using the algorithm described in Section 6.2.2.

These two strategies are adaptations of the ones that were first proposed and studied for disjunctive atomic queries, and the more general preference relations expressed as SPOs in [120]. Here, I adapt the techniques to the special case of score functions, and the general case of UNCQs.

7.2.1 Aggregation-last Query Answering

Let us start by formalising the notion of top-k aggregation-last answers. Given a Datalog$^{\pm}$ ontology O, a set of n users $\mathcal{U} = \{u_1, \ldots, u_n\}$ along with score functions $score_{u_1}, \ldots, score_{u_n}$, a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, and an aggregation operator $\uplus$, in the following, I denote with $k\text{-}ans_i$ the set $ans_k(q, (O, score_{u_i}))$, i.e., the top-$k$ answers for user u_i.

Note that $k\text{-}ans_i$ consists of at most k pairs of the form $\langle t, scr_t \rangle$, where $t \in D$, and scr_t is as in Definition 6.2.5. Intuitively, user u_i is inducing a score-based preference relation among the tuples in D, where the score for every tuple not appearing in $k\text{-}ans_i$ is zero; therefore, in the following definition, I use $k\text{-}ans_i$ to represent the set of top-k answers for user u_i as well as to represent the score-based preference relation that it induces.

Definition 7.2.1 (**Agg-last Top-k Answers**)**.** Given a Datalog+/– ontology O, a set of n users $\mathcal{U} = \{u_1, \ldots, u_n\}$ with corresponding score functions $score_{u_1}, \ldots, score_{u_n}$, a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, and an aggregation operator $\uplus$, let $A = \{\langle t, scr_{t,u_i} \rangle \mid \langle t, scr_{t,u_i} \rangle \in \bigcup_{u_i \in \mathcal{U}} k\text{-}ans_i\}$ and $TU = \{t \mid \exists \langle t, scr \rangle \in A\}$. An *Agg-Last* top-$k$ answer for $\mathcal{U}$ is a set $ansG\text{-}last_k(q, O, \mathcal{U})$ such that:

(i) $ansG\text{-}last_k(q, O, \mathcal{U}) \subseteq A$,

(ii) for each $\langle t, score^*(t)\rangle \in ansG\text{-}last_k(q, O, \mathcal{U})$, we have $score^*(t) \geq score^*(t')$ for every tuple t' appearing in A but not in $ansG\text{-}last_k(q, O, \mathcal{U})$, where $score^* = \biguplus(TU, k\text{-}ans_1, \ldots, k\text{-}ans_n)$, and

(iii) $|ansG\text{-}last_k(q, O, \mathcal{U})| = min(k, |TU|)$.

Let us now study how to compute Agg-Last top-k answers for a set of users by leveraging Algorithm RankJoinUNCQ described in Algorithm 11. The main strategy works as follows: (i) for each user u_i, compute the top-k answers to query $q(\mathbf{X})$ using Algorithm RankJoinUNCQ; and (ii) aggregate the answers of all users into one top-k answer using a score aggregation operator $\biguplus$. Algorithm AggLast, whose pseudocode is shown in Algorithm 19, performs this process. Clearly, in step (i), RankJoinUNCQ could be replaced by any algorithm that computes top-k answers for preference-based Datalog$^{\pm}$ ontologies. However, all the results shown in this chapter assume the use of Algorithm RankJoinUNCQ.

Algorithm 19 Agg-Last top-k query answering.

Input: (i) Datalog$^{\pm}$ ontology O,
(ii) set of users $\mathcal{U} = \{u_1, \ldots, u_n\}$, where each u_i has associated score function $score_{u_i}$,
(iii) UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$,
(iv) integer $k > 0$, and
(v) score aggregation operator $\biguplus$.
Output: A top-k answer to $q(\mathbf{X})$.
Calls: Algorithm 11.
Global Variables: *relList*, *hashTab*, *Its*, a set TS of pairs $\langle t, p\rangle$.

```
1: function AggLast(O, U, q, k, ⊎)
2:     for all users u_i ∈ U do
3:         k-ans_i ← RankJoinUNCQ((O, score_{u_i}), q(X), k); //Algorithm 11
4:     Res ← ⊎(k-ans_1, ..., k-ans_n);
5:     return k highest-scoring elements in Res.
```

The algorithm first computes a top-k answer to query $q(\mathbf{X})$ for each user—Algorithm RankJoinUNCQ returns a set of k pairs $\langle t, scr_t\rangle$ for each user; then, in line 3, the aggregation operator is applied to these results.

In the following, I consider the following aggregation operators based on *ranking aggregation methods* [143]: $\biguplus_x$ with $x \in \{max, min, borda, sum, avg\}$. For every tuple t in R, operator $\biguplus_{max}(R, score_1, \ldots, score_n)$ assigns to t the maximum score among all $score_i(t)$. Operators $\biguplus_{min}$, $\biguplus_{avg}$, and $\biguplus_{sum}$ apply minimum, average and sum, respectively. Clearly, $\biguplus_{max}$ (resp. $\biguplus_{min}$) is equivalent to the max-based (min-based) linear combinator mentioned in [143]. For $\biguplus_{borda}$ (Borda count in [143]), each user ranks their top-k answers. For each user, the top-ranked answer is given k points, the second-ranked one $k-1$ points, and so on. Note that one first needs to normalise the scores in each individual rank, as shown in [143].

Example 7.2.1. Consider again query $q_1(U, P)$ from Example 7.1.1. First, the top-1 answer for each user is computed using RankJoinUNCQ. We have then that the top-1 answer for u_1 is $k\text{-}ans_1 = \{\langle(b, p_1), 0.6\rangle\}$ (as computed in Example 6.2.2); possible top-1 answers for u_2 and u_3 are $k\text{-}ans_2 = \{\langle(b, p_4), 0.5\rangle\}$ and $k\text{-}ans_3 = \{\langle(b, p_4), 0.7\rangle\}$, respectively. The universe of elements in this case (i.e., the set TU in Definition 7.2.1) is $\{(b, p_1), (b, p_4)\}$. The operator $\biguplus_{max}$ yields $\langle(b, p_1), 0.6\rangle$ and $\langle(b, p_4), 0.7\rangle$; thus, the top-1 answer for the group is $\langle(b, p_4), 0.7\rangle$ using $\biguplus_{max}$. ∎

An alternative to the aggregation operator is to consider voting mechanisms from the social choice literature (as proposed in [120]). For example, $\biguplus_{plurality}$ can be used, which works as follows: one first computes the individual top-k answers for each user (a user's top-k choices are taken as their top-preferred items). Then, each items' frequency for all the users are summed up, and the k items with the highest number of votes win. The final scores in this case consist of assignments of 1 to the chosen tuples and 0 to the rest.

Example 7.2.2. If we use the operator $\biguplus_{plurality}$ in Example 7.2.1, then (p_1, f_4) has two votes, and element (b, p_1) has one vote; therefore, the same answer as before is obtained through plurality voting. ∎

Proposition 7.2.1. Given a Datalog$^{\pm}$ ontology $KB = (D, \emptyset)$, a set of users $\mathcal{U}$, a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, and an aggregation operator $\biguplus$, Algorithm AggLast correctly computes an *Agg-Last* top-k answer for $\mathcal{U}$.

7.2.2 Aggregation-first Query Answering

Under the aggregation-first approach, the preferences of all users are taken into account in the generation of a new score function that somehow encodes the preferences of the group as a whole. This single-user preference function is then used to compute the top-k answers to queries. Let us formalise this in the following definition.

Definition 7.2.2 (**Agg-first Top-k Answers**)**.** Given a relational schema $\mathcal{R}$ and a Datalog$^{\pm}$ ontology O on $\mathcal{R}$, a set of n users $\mathcal{U} = \{u_1, \ldots, u_n\}$ with corresponding score functions $score_{u_1}, \ldots, score_{u_n}$, a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, and an aggregation operator $\biguplus$, an *Agg-First top-k answer* for $\mathcal{U}$ is the set $ansG\text{-}first_k(q, O, \mathcal{U}) = ans_k(q, KB')$ where $KB' = (KB, score^*)$ is a preference-based Datalog$^{\pm}$ ontology and $score^* = \biguplus(R, score_{u_1}, \ldots, score_{u_n})$ for every $R \in \mathcal{R}$.

Algorithm 20 implements Agg-First top-k answering using RankJoinUNCQ (Algorithm 11) to compute the top-k answers to a UNCQ based on the aggregated preference relation. Analogously, Algorithm 19, the call to RankJoinUNCQ in line 4 can be replaced by one to any algorithm that computes the top-k answers for a preference-based Datalog$^{\pm}$ ontology.

Example 7.2.3. Consider again CQ $q_1(U, P)$ from Example 7.1.1. The first step in AggFirst is to aggregate the scores of the different users in every relation appearing in the query. In this case, we have relations *property*, *user*, and *offer*. The following table summarises the result of aggregating the individual scores determined by score functions $score_{u_1}$, $score_{u_2}$, and $score_{u_3}$ for each tuple in these relations. We assume aggregation operator $\biguplus_{max}$ as in Example 7.2.1.

property	
tuple	$score^*$
t_4	0.9
t_1	0.8
t_2	0.7
t_3	0.6
t_5	0.5
t_6	0.0

offer	
tuple	$score^*$
t_{16}	0.9
t_{14}	0.7
t_{17}	0.65
t_{15}	0.6

user	
tuple	$score^*$
t_7	0.8
t_8	0.5
t_9	0.2

After the aggregation step, Algorithm RankJoinUNCQ is called for q_1 as before, but function $score^*$ is used instead. Note that Algorithm RankJoinUNCQ assumes that the tables appearing in q_1 are sorted by $score^*$ in descending order. The top-1 answer in this case is $\{\langle(b, p_4), 0.8\rangle\}$. ∎

Algorithm 20 Agg-First top-k query answering.

Input: (i) Datalog$^{\pm}$ ontology O,
(ii) set of users $\mathcal{U} = \{u_1, \ldots, u_n\}$, where each u_i has associated score function $score_{u_i}$,
(iii) UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$,
(iv) integer $k > 0$, and
(v) score aggregation operator $\biguplus$.
Output: A top-k answer to $q(\mathbf{X})$.
Calls: Algorithm 11.
Global Variables: *relList*, *hashTab*, *Its*, a set TS of pairs $\langle t, p\rangle$.

```
1: function AggFirst(O, U, q, k, ⊎)
2:     Let qR be all relations that appear in q(X);
3:     for all R_i ∈ qR do
4:         Res_i := ⊎(R_i, score_{u_1}, ..., score_{u_n});
5:         for all t ∈ R_i do
6:             score*(t) ← Res_i(t);
7:     return RankJoinUNCQ((O, score*), q(X), k). //Algorithm 11.
```

Proposition 7.2.2. Given a Datalog$^{\pm}$ ontology $KB = (D, \emptyset)$, a set of users $\mathcal{U}$, a UNCQ $q(\mathbf{X}) = \bigvee_{i=1}^{l} q_i(\mathbf{X})$, and an aggregation operator $\biguplus$, Algorithm AggFirst

correctly computes an *Agg-First* top-k answer for $\mathcal{U}$.

Note that, in certain cases, it may not be necessary to produce the aggregated scores for the entire tables used in the query; instead, I compute the required tuples on demand to avoid unnecessary computations. The specific strategy to accomplish this depends on the aggregation operator used, and may not be possible in general. $\biguplus_{max}$ is one example of an operator, for which relatively minor changes to my algorithm suffice to implement this optimisation.

7.3 Semantic Properties

An interesting line of research is the study of the different properties that are satisfied (or violated) by the different approaches that address top-k query answering. Several works in the literature study semantic properties for top-k answers over certain and uncertain data; among them we refer the reader to [44, 178]. As we intend to model the preference relation that represents the preferences of a group of individuals as a whole, we have focused instead on a set of properties that are based on the ones usually studied in social choice theory [69]; such properties were studied in [120] for strict partial orders (that, for their generality, also offer less guarantees). Nevertheless, it is important to note that most of the properties that can be found in [44, 178] are subsumed and adapted for the group preference scenario.

7.3.1 Aggregation-last Top-k Querying

An in-depth study of such properties is out of the scope of this chapter; however, we present a brief discussion to show the kinds of results that we have obtained so far and are pursuing further. The Monotonicity and Stability properties are adapted from my previous work on answering top-k queries over groups in a more general setting [120].

The following two properties refer to the effects of improving and lowering the scores of tuples. We denote as $Ans_k(q, KB)$ the set containing all the top-k answers

$ans_k(q, KB)$.

Monotonicity 1: Given $k > 0$, let $r \in Ans_k(q, KB)$ for $\mathcal{U} = \{u_1, \ldots, u_n\}$, $t \in r$, and $u_i \in \mathcal{U}$ be such that $score_{u_i}(t) = v$. If $u_i' \notin \mathcal{U}$ is identical to u_i except that $score_{u_i}(t) = v'$ with $v' > v$; then, there exists $r' \in Ans_k(q, KB)$ for $\mathcal{U}' = \{u_1, \ldots, u_{i-1}, u_i', u_{i+1}, \ldots, u_n\}$ such that $t \in r'$.

Intuitively, Monotonicity 1 states that if a tuple t is in a top-k answer, it is still in a top-k answer if the score function of some user changes so that t's (and only its) score is increased.

Monotonicity 2: Given $k > 0$, let t be a tuple such that there is no $r \in Ans_k(q, KB)$ for $\mathcal{U} = \{u_1, \ldots, u_n\}$ with $t \in r$, and let $u_i \in \mathcal{U}$ be such that $score_{u_i}(t) = v$. If $u_i' \notin \mathcal{U}$ is identical to u_i except that $score_{u_i}(t) = v'$ with $v' < v$; then, there does not exist $r' \in Ans_n(q, KB)$ for $\mathcal{U}' = \{u_1, \ldots, u_{i-1}, u_i', u_{i+1}, \ldots, u_n\}$ such that $t \in r'$.

Conversely, Monotonicity 2 states that if a tuple t does not appear in any answer, it cannot appear in any answer if the score function of a user changes so that t's (and only its) score is decreased.

The next property is concerned with what happens when a new tuple c to choose from is added. In the following, I use notation KB_{add} to refer to the knowledge base that results from adding this new element, and assume that all score functions are extended accordingly.

Faithfulness: Given $k > 0$, let a and b be two tuples, such that for all $u \in \mathcal{U} = \{u_1, \ldots, u_n\}$ it holds that $score_u(a) > score_u(b)$; then, there is no top-k answer $r \in Ans_k(q, KB)$ for $\mathcal{U}$ such that $pos(b, r) > pos(a, r)$.

This property, adapted from [179], states that if every user assigns a greater value to tuple a than tuple b, then a must appear before b in every possible top-k answer.

Proposition 7.3.1. Algorithm AggLast satisfies:

1. Monotonicity 1 and Monotonicity 2, for operators $\biguplus_x$ with $x \in \{plurality, max, sum, avg\}$; and

2. Faithfulness for operators $\biguplus_x$ with $x \in \{max, sum, avg\}$.

Algorithm AggLast satisfies:

1. Monotonicity 1 and Monotonicity 2, for operators $\biguplus_x$ with $x \in \{plurality, max, sum, avg\}$; and

2. Faithfulness for operators $\biguplus_x$ with $x \in \{max, sum, avg\}$.

Proof (sketch).

Monotonicity 1: Consider AggLast and $\biguplus_{plurality}$, and let $k\text{-}ans_i$ be top-k answers to q for each u_i; since only the score function for u_i changes, then we can assume that the same other $n-1$ answers are obtained when computing the top-k answer for each user in $\mathcal{U}'$. Clearly, all atoms appearing in $k\text{-}ans_i$ maintain the same number of votes, including t; therefore, t remains in the top-k answer for $\mathcal{U}$.

Consider AggLast and $\biguplus_{max}$, and let $scr_{\mathcal{U}}(t)$ be the score for t in answer r; this score is computed as the maximum among all the scores assigned to t in all individual top-k answers. Then, as only the score of t changes in $k\text{-}ans'_i$, we have that $scr_{\mathcal{U}'}(t)$, the score for t in answer r', is greater than $scr_{\mathcal{U}}(t)$; as the scores for the rest of the answers remain the same, t must remain within the top-k answer. An analogous argument holds for $\biguplus_{avg}$ and $\biguplus_{sum}$.

Monotonicity 2: Consider again the argument for Monotonicity 1 and $\biguplus_{plurality}$; clearly, as t received no votes prior to the change in score, it will not receive any votes after, and thus cannot appear in any top-k answer. For $\biguplus_{max}$, $\biguplus_{avg}$, and $\biguplus_{sum}$, a similar argument can be made—if t did not influence the value of max, avg, or sum before the change to make the element appear in a top-k answer, it cannot do so after its score is lowered even further.

Faithfulness: First, I clarify that this property is not satisfied by $\biguplus_{plurality}$ because of the way votes are assigned: it doesn't matter that all users score element a higher than b—as long as both elements are in individual users' top-k answers, they both

receive one vote. Thus, ties in number of votes leads to the existence of top-k answers that exchange the positions of these elements.

For *max*, *avg*, and *sum*, a similar argument to the one used for Monotonicity can be applied; these functions enjoy the uniformity property, so if $score_{u_i}(a) > score_{u_i}(b)$ for all $u_i \in \mathcal{U}$, then the aggregate function applied to each side preserve the relation. □

7.3.2 Comparison with Previous Work

It is interesting to contemplate the relationship between the semantic properties that hold in the more general models of [120, Fig. 6] and the score function-based ones in this chapter. Of course, the positive results obtained there carry over to this setting; unfortunately, some of the negative results also carry over. One interesting property is *Stability*, which is included in [120, Fig. 6] as two separate properties, one in the case in which a new element is added and the other in which an existing element is removed. Let us reformulate the former for the present model:

> *Stability-Add*: For each top-k answer $r \in Ans_k(q, KB)$ for $\mathcal{U} = \{u_1, \ldots, u_n\}$, there exists a top-$k$ answer $r' \in Ans_k(q, KB_{add})$ such that either $r = r'$ or (i) $r' - r = \{c\}$ and (ii) let $r = \langle a_1, \ldots, a_k \rangle$ and $r' = \langle b_1, \ldots, b_k \rangle$; then, for each pair b_i, b_j such that $1 \leq i < j \leq k$ and $b_i = a_{i'}$ and $b_j = a_{j'}$, it holds that $i' < j'$.

Essentially, the property ensures that if a new element is added, the result is either unaltered or the new element appears, and the relative order of all the rest stays the same. This property is not satisfied in the general SPO-based model, but one could hope that the added structure of scores would help (at least for some operators). Unfortunately, it is simple to construct counterexamples for the operators studied in this chapter. For $\biguplus_{plurality}$, this is accomplished by adding element c in such a way that other elements are pushed out of the individual top-k answers, causing the number of votes to change in such a way that the final order is altered. The

same kind of counterexample is possible for the cases of *avg* and *sum*. For $\biguplus_{max}$, the same kind of strategy leads to a counterexample for the property; for the property to be violated, there must exist two elements, a and b, such that they are swapped in the top-k answer after the addition of the new element. That is, $scr(a) > scr(b)$ and $scr'(b) > scr'(a)$, where $scr(.)$ and $scr'(.)$ denote the scores associated with the element in the answer prior to the addition and after, respectively. Suppose now that a loses its position (to c) in an individual top-k answer; if that was the score that determined $scr(a)$, then clearly $scr'(a)$ can be lower than $scr'(b)$.

Other negative results explored in previous work still hold here, with the same counterexamples found for the more general model. These include the other version of the Stability property for Plurality, called S2 in [120] (where an element is removed), as well as those for Fairness (Unanimous Winner, both Stability properties, and Non-Dictatorship when $k = 1$). We refer the interested reader to [120, Appendix B.5] for the counterexamples.

The properties discussed here are most naturally formulated for the aggregation-last approach; for aggregation-first, the relationship between the scores given to basic tuples by each user and the final answers to queries is not as clear—as future work we would like to see what properties can be used to compare aggregation-first approaches in a principled manner.

7.4 Implementation and Evaluation

Let us now report the results of evaluating my algorithms on real-world data, analysing both running time and quality of their results.

7.4.1 Implementation and Hardware

I implemented the algorithms by extending the Datalog+/– query answering engine [77], which involved adding query answering with score-based preferences for a group of users. I implemented four algorithms for AggLast and two for AggFirst—the

code and dataset will be made available as open source. For AggLast, these algorithms are LastBorda, LastPlurality, LastHitMax, and LastHitMin, which use $\biguplus_x$ with x among *borda*, *plurality*, *max*, and *min*, respectively. I use score normalisation [143] for LastHitMax and LastHitMin, and Borda normalisation [143] for Borda. To compute the final scores, I used linear combination: the sum of normalised Borda scores for Borda, and the sum of hits (the number of users that include an element in their individual result) multiplied by max (resp., min) of score normalisation for LastHitMax (resp., LastHitMin). For AggFirst, I implemented FirstAvg and FirstMax, which use $\biguplus_x$ with x among *avg* and *max*, respectively.

The entire implementation was done in Java; all runs were done on an Intel Core i7 processor at 2.2 GHz and 8GB of RAM, under the MacOS X 10.6.8 OS, and a Sun JVM Standard Edition with maximum heap size of 512 MB. To minimise experimental variation, all results are averages of three independent runs.

7.4.2 Experimental Setup

Inputs to my system consist of tuples $(q, O, \mathcal{U}, k)$, where q is a query, O is the Datalog$^{\pm}$ ontology, $\mathcal{U}$ is a group preference model, and k is the number of query results.

Data. All runs were carried out using an ontology built on data from the Yelp Dataset Challenge [175], which contains 11,537 businesses in the Phoenix (USA) metropolitan area, 8,282 check-ins, 43,873 users, and 229,907 reviews—each business has one or more associated categories. The Datalog$^{\pm}$ ontology was constructed as follows. I created seven different relations: *place*, *placeType*, *cuisine*, *food*, *isPlaceType*, *servesCuisine*, and *servesFood*; e.g., given the "sushi-bar", "Asian", and "sushi" categories for a business x, called *name* in city y, we have $place(x, y, name)$, $placeType(sushi\text{-}bar)$, $cuisine(asian)$, $food(sushi)$, $isPlaceType(x, sushi\text{-}bar)$, $servesCuisine(x, asian)$, and $servesFood(x, sushi)$.

User Preferences. I used the preference dataset[1] gathered in previous work [120], which consists of preferences for 49 users over *cuisine*, *type of food*, and *type of place* (breakfast, lunch, and dinner). Users entered their preferences as strict partial orders (represented as graphs) via a GUI. For each such graph G, I computed the *layer* corresponding to each vertex (the undominated nodes comprise layer 1, and so on). If the number of layers of G is n, and vertex v belongs to layer ℓ, then the score of v is computed as a random number in $[1-(\ell/n), 1-(\ell-1/n)]$ (e.g., if the users specified that they prefer bagels over sushi, then the score of *food*(*bagel*) is higher than that of *food*(*sushi*)).

For each business, the Yelp dataset provides a numerical rating from 0 to 5. I used this information to compute the scores of tuples of the relation *place* as a random number in the interval $[(r-\sigma)/(5+\sigma), (r+\sigma)/(5+\sigma)]$, where r is the rating, and σ is the associated standard deviation. To compute the scores for the ground atoms $\mathit{isPlaceType}(p,t)$, $\mathit{servesCuisine}(p,c)$, and $\mathit{servesFood}(p,f)$, I computed the max between the score of $\mathit{place}(p,y,\mathit{name})$ and the scores of $\mathit{placeType}(t)$, $\mathit{cuisine}(c)$, and $\mathit{food}(f)$, respectively.

Group Definition. I used the same groups from [120]: 19 groups of 3 to 9 users. This produced an overall number of $19 \cdot 3 = 57$ group choice scenarios.

Queries. I used the following UNCQs:

$$q(X) = \exists F, C, T\,(q_1(X,F) \vee q_2(X,C) \vee q_3(X,T)),$$

$$q'(X) = \exists F, C, T\,(q_4(X,F) \vee q_5(X,C) \vee q_3(X,T)), \text{ and}$$

$$q''(X) = \exists F, C, T\,(q_1(X,F) \vee q_2(X,C) \vee q_6(X,T)), \text{ where}$$

$$q_1(P,F) = \mathit{food}(F) \wedge \neg \mathit{servesFood}(P,F) \wedge \mathit{place}(P),$$

$$q_2(P,C) = \mathit{servesCuisine}(X,C) \wedge \mathit{cuisine}(C) \wedge \mathit{place}(P),$$

$$q_3(P,T) = \mathit{placeType}(T) \wedge \mathit{isPlaceType}(X,T) \wedge \mathit{place}(P),$$

$$q_4(P,F) = \mathit{food}(F) \wedge \mathit{servesFood}(X,F) \wedge \mathit{place}(P),$$

$$q_5(P,C) = \neg \mathit{servesCuisine}(X,C) \wedge \mathit{cuisine}(C) \wedge \mathit{place}(P),$$

[1] https://github.com/personalised-semantic-search/dataset_qualitative

Figure 7.1: Running time performance.

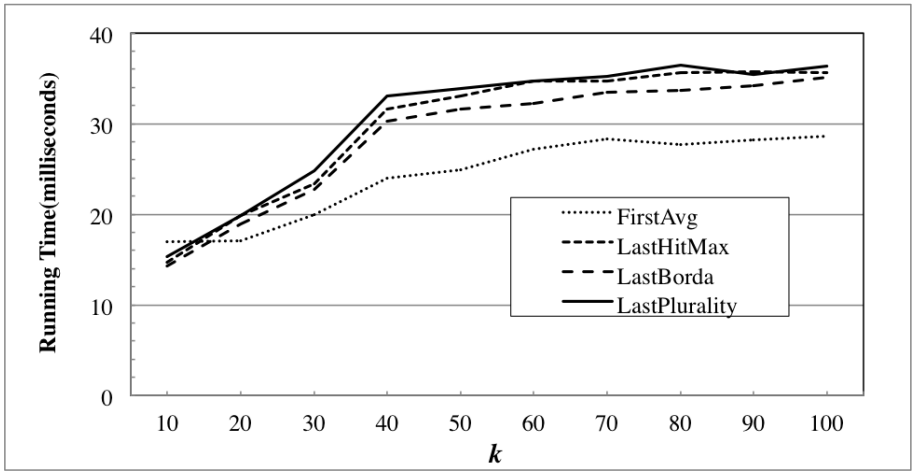
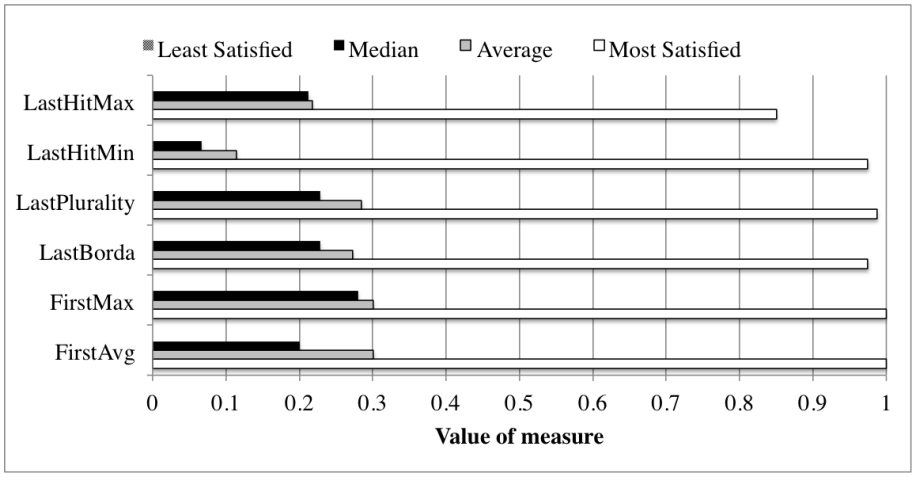


Figure 7.2: Quality evaluation: Agreement



$q_6(P,T) = placeType(T) \wedge \neg isPlaceType(X,T) \wedge place(P)$.

These queries represent situations where groups wish to decide where to go for a meal. For instance, q requests places where favourite cuisines are served, or the place type is a favourite, or the place does not serve a favourite food. All runs have $10 \leq k \leq 100$, varied in steps of 10.

7.4.3 Results

Performance Evaluation. For the performance metric, I use the time required to answer queries, varying k. Figure 7.1 reports the result of this evaluation: FirstAvg is faster than the other methods for this dataset, especially at higher values of k. The

Figure 7.3: Quality evaluation: Other Metrics

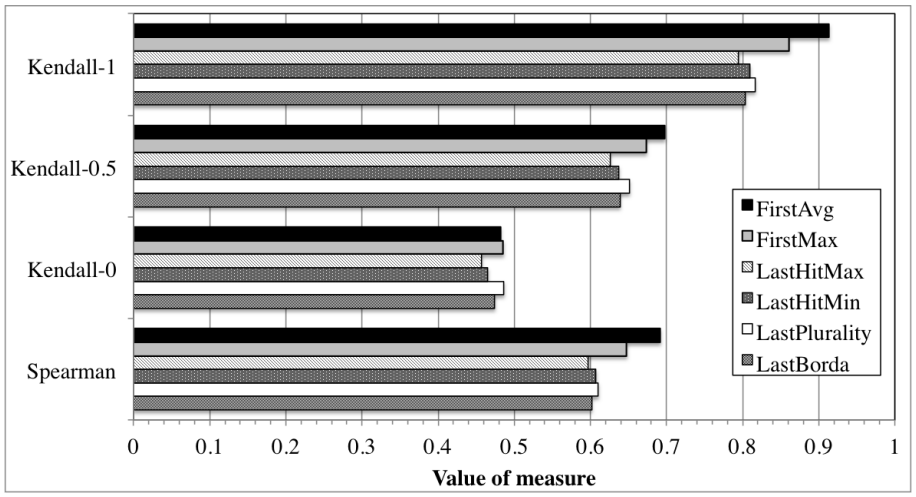
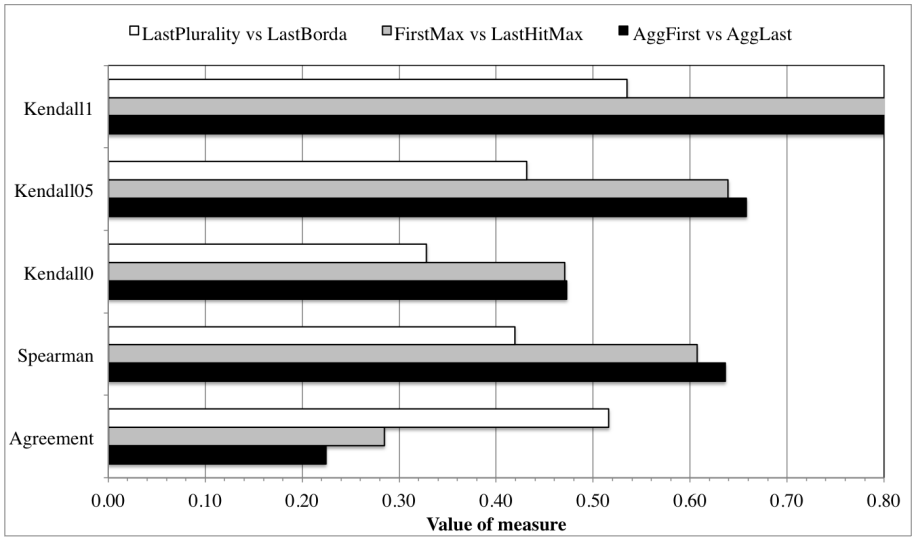


Figure 7.4: Method comparison



evaluation of the materialisation is not considered here (I computed the materialised ontology off-line); the performance of materialisation with different benchmarks has already been shown to be practical in terms of the time, size, and memory consumption for the engine that I extend [77]. Figure 7.1 shows that my algorithms are very efficient: even the slowest ones return a top-k answer in less than 40 ms for high values of k, confirming the feasibility of my approach.

Quality Evaluation. Let us now analyse the results of the quality evaluation of my algorithms.

Evaluation Metrics. I use quality measurements that are often applied in evaluating

information retrieval and group recommender systems [62,135]; the main ones adopted are *tuple agreement*, *Kendall tau distance*, and *Spearman's footrule*.

Given two top-k lists, *tuple agreement* is defined as the number of tuples that appear in both lists (ordering is not considered). To allow comparisons across different values of k, I normalise the tuple agreement and call this value *Agreement—higher is better*, a value of 1 indicating lists with the same elements. The second measure is a variation of the *Kendall tau distance* for partial orders that computes the distance between two partial rankings based on the number of pairwise disagreements between them [62]—a disagreement receives a penalty of 1. In case the two top-k lists are permutations of each other, this is the number of exchanges required to convert one into the other. for cases where a pair (i, j) appears in one list but not in the other, it takes a parameter p indicating how pessimistic the metric should be ($p = 1$ is most pessimistic). The measure is normalised by the square of the length of the union of the two lists, and this value is called *Kendall*$_p$*—lower is better for Kendall*. Finally, the third measure is a variation of *Spearman's footrule* for partial orders that computes the distance between two partial rankings using the difference of positions of elements of one list compared with the other [62]. Whereas *Kendall* counts number of swaps, this metric counts how far each element must be moved to reach the place occupied in the other list. This measure is normalised as before, and the result is called *Spearman—lower is better for Spearman*.

Question 1: Which aggregation method yields the best results? To answer this question, for each group I computed a top-k query answer r_g, along with a top-k query answer r_i for each user in the group. I then computed the value of each measure over r_g and r_i, and finally report the result of aggregating all such results. The choice of aggregation function is dependent on what is considered to be *best*: the value of the least satisfied user's score (min for Agreement, max for the others), the most satisfied user's score (max for Agreement, min for the others), the median satisfaction, or the mean satisfaction. I computed these four functions for the Agreement measure; Figure 7.2

shows that on average AggFirst (FirstAvg and FirstMax) is better than AggLast (LastBorda, LastPlurality, LastHitMax, and LastHitMin). Since there are cases where some user did not have any of their items in the top-k of the group for any of the methods, the least satisfied users have $Agreement = 0$ for all methods (this is why least satisfied is not plotted). In case of LastBorda and LastPlurality, some users have all their items in the top-k in the group, while for other methods this does not hold. For reasons of space, in the following I only show the results for the average. Figure 7.3 shows that LastHitMax is the best at keeping the ordering of the items in users' top-k and the items in the group's top-k. From this point of view, FirstAvg is the worst method.

Question 2: How different are the results produced by each aggregation method? Figure 7.4 plots how different approaches compare when taken pairwise; "AggFirst" and "AggLast" are aggregations of methods computed as means over pairs (a_1, a_2), where a_1 is an aggregation first method and a_2 is an aggregation last. We can see that AggFirst and AggLast are more similar when considering the right ordering than they are with respect to the Agreement measure. LastPlurality and LastBorda seem to include more elements in common and similar ordering in comparison with the AggFirst–AggLast pair and FirstMax–LastHitMax pair. Since the choice of the conjunctive query and of the relation in the *chooseRel* subroutine is done randomly in my implementation, this explains the fact that we do not observe high Agreement between methods and users: the top-k converges, but the list of elements that are higher or equal to the minimum score of top-k can be larger than k.

Finally, I ran two-tailed two-sample Student's t-tests comparing all pairs of experiments reported in this section. All tests yielded p-values well below 0.001, which shows statistical significance, except for three cases: (i) "LastPlurality vs. User"/ "Borda vs. User" for Agreement ($p = 0.84$) and Spearman ($p = 0.01$), (ii) "Borda vs. User" / "LastHitMin vs. User" for Spearman ($p = 0.18$) and Kendall-1 ($p = 0.28$), Kendall-0.5 ($p = 0.56$), and (iii) "LastHitMin vs. User" / "LastHitMax

vs. User" for Agreement ($p = 0.05$), Spearman, Kendall-0.5 and Kendall-1 ($p = 0.01$ in all cases)—note that these higher p-values are expected, given the similarity of the results reported in those cases.

7.5 Conclusion

This chapter has proposed an approach to top-k query answering under user preferences in Datalog$^{\pm}$ ontologies, where the queries are unions of conjunctive queries with safe negation, and the preferences are defined via numerical scores. To this end, I have generalised the previous RankJoin operator to my framework, and to the preferences of a group of users. Finally, I have provided experimental results on the performance and quality of my algorithms.

One topic of ongoing and future work is to further experimentally evaluate the similarity of preference aggregations to human judgement in order to select the best suited ones for search and query answering in the Social Semantic Web. Another interesting topic for future research is to generalise the presented approach to ontologies with uncertainty.

Part V

Discussion

Chapter 8

Literature Review

This chapter reviews prior work in area of preference handling. Literature review related with preferences in AI . This thesis is mostly concerned with the problem of preference modelling and reasoning in Semantic Web, therefore I discuss different preference representation languages and the work done in preference modelling for databases (where I inspired some of my work from) and Semantic Web.

Preferences have long been studied in many disciplines, prominently in philosophy, databases, and AI (for literature reviews [53, 92, 140]). In philosophy, the research mainly deals with *preference logics*, where preferences are usually expressed over mutually exclusive worlds like truth assignments to formulas, and the research focus lies on axiomatizations.

8.1 Preference Languages

In this section I analyse some of the preference representation languages that I drew my inspiration from.

8.1.1 Quantitative Preference Languages

Utility functions [163] assign to outcomes a real numbers, such that the higher the number, the more preferred is the outcome. The *possibility theory* [176] assigns a numerical value between [0,1], that shows the importance of the preference formula

for the user. If the weight is 1 than the formula must be satisfied. When computing outcomes weight, the focus is on formulas having the greatest weight. *Possibilistic logic* [55] is weaker than penalty logic, with the sum operator being replaced by min operator (penalty logic considers all formulas while possiblistic logic considers only falsified formulas with biggest weight). The possibility theory with bipolar preferences has been investigated in [12].

Penalty logic [49] the preferences are constructed using a finite set of weighted propositional logic formulas. The weight represents the penalty of falsifying the preference formula. The weight of the outcome o is computed by adding up all the penalties over the preferences formulas that do not satisfy o. If all the preferences formulas satisfy o, than the penalty is 0.

The extension of lexicographic ordering for score based preferences [172] considers that each features has an associated vector of costs. It is assumed that there is an importance ordering on criteria, and that to compare two outcomes, firstly, the combined cost of each outcome with respect to the most important criteria is compared; only if these combined costs are equal, the next most important criteria is considered.

With respect to probabilistic preferences, there are several works that have been developed in the last few years. Probabilistic skylines were introduced in [138] (and later also studied in [4]; the related *stochastic* skyline is introduced in [109]), where the authors tackle the problem of computing the probability of an element belonging to the skyline, rather than using the probability values for the purpose of ranking, as proposed in this thesis. In [150, 179], the authors focus on a more specific version of my problem, since they assume that tuples receive both a score and a probability value, and thus the two preference relations in question are score-based; furthermore, the approach adopted in their work is based on possible worlds, whereas, here, I focus on the use of probabilities solely as vehicles for ranking.

8.1.2 Graphical and Conditional languages

Generalised Additive Independence Networks (GAI-nets) [73] are graphical languages that express preferences using weights. The undirected graph consists of factors (that are subsets of attributes such that their union represents the attribute set) as nodes. The variables in a factor are assumed to be independent, and each factor has associated a weight. The overall utility is computed by adding up the utility of he factors.

CP-nets [21] (one of the most widely known formalisms), *TCP-nets* [25] are graphical languages that are subsets of *CP-theories* [168]. *TUP-nets* [146, 147] is an unconditional fragment of TCP-nets (can express unconditional intra-variable preferences, unconditional relative importance preferences), with dominance testing computed in polynomial time. CP-nets can can express preference for exactly one value over another while General CP-nets (GCP-nets) [72] can have incomplete or locally inconsistent CP-nets. Most work on CP-nets has focused on the computation of optimal outcomes and dominance testing, i.e., to check if one outcome of the CP-net is preferred to another. More recently, [165] [[165]] propose an efficient algorithm and indexing scheme for top-k retrieval in CP-nets. CP-nets are extended with hard and soft constraints [54] while *Conditional preference networks with utilities*(UCP-nets) [19] are a language that lies between CP-nets and GAI-nets

Conditional importance (CI) nets [22] compares sets of variables (e.g., "if I have a set A of goods, and I do not have any of the goods from some other set B, then I prefer the set of goods C over the set of goods D") for the description of fair division problems.

Comparative preference theories [94, 170, 171] have preferences defined on a set of features simultaneously. V is the variable set, $P \subseteq V, Q \subseteq V, T \subseteq V$ and p is an assignment to P , and q is an assignment to Q. The statements are of the form: $p > q||T$ that represent "p is preferred to q if T is held constant". They define a special kind of weak order generated by what they call a CP-tree, that is general

form of lexicographic order, but where the importance ordering on variables can depend on more important variables, as can the value orderings.

Qualitative logic [27] adds to propositional logic preferences over alternatives "if possible A but if not B".

8.2 Preferences in Databases

Modelling and dealing with preferences in databases has been studied for almost three decades, since the seminal work of [103] [[103]] which extends the relational calculus with preference modelling mechanisms for query answering; see [41, 112, 151] for a survey of notable works in this line.

Preference formulas (e.g., "tuple t_1 is preferred t_2 some conditions hold") are introduced as a logical formalism [39] that allows an embedding of preference specifications into SQL through a *winnow* operator parametrized by a preference formula; the winnow operator is a generalisation of the *skyline* operator, first introduced in [16] and optimised in [158] (also known as *Best Match Only* operator [95]). Perhaps closest to my approaches is that of preference Datalog programs [80], which are a restriction of preference logic programs [81] that contain no uninterpreted function symbols and extend classical Datalog with constructs for determining which predicates must be optimised along with the optimisation criteria (i.e., the set of preferences).

Ranking queries have many applications in information systems: scores assigned to data can have many meanings, such as desirability to users, similarity to objects of interest, or quantifications of uncertainty associated with pieces of information—in all these cases, obtaining the top-k elements that satisfy a certain condition is a basic operation, and carrying it out efficiently is thus of central interest. In databases, this was recognised over a decade ago with the incorporation of tuple ranking into the RDBMS at the same basic level as Boolean filtering is implemented [85, 106]. This line of work builds on prior research on the basic problem of computing ranked

answers to queries in relational databases [59, 63, 131]; for a relatively current survey, see [86] and the more recent work of [3].

Databases research have also modelled preferences using CP-nets [24], incomplete CP-nets [42], TCOP-nets [57],conditional preferences [47], conditional preference relational networks using ceteris paribus semantics [98], conditional preferences that generalises GCP-nets and CP-theories that uses Datalog [45].

Work has also been carried out in the intersection with databases and knowledge representation and reasoning, such as preference logic programs [81], incorporation of preferences into formalisms such as answer set programs [26].

8.3 Preferences and the Semantic Web

As my main interest in the problem arises from Semantic Web applications, perhaps closest to my work is that of [152], which explores top-k query answering over relational databases that are accessed via *DL-Lite* ontologies (i.e., ontology-mediated access). The main differences with my approaches lie in the query language (they consider only conjunctive queries (without negation)) and the ontology language (guarded Datalog$^{\pm}$ subsumes the entire *DL-Lite* family of Description Logics). Moreover, their approach is based on query rewriting rather than database materialisation.

The preference formalism that combines CP-nets and DLs in [51], where variable values of CP-nets are satisfiable DL formulas. The main difference with my work lies in the use of CP-theories here rather than CP-nets, and (more importantly) in the relationship between the ontology and the CP-net/theory: they use ontological axioms to restrict CP-net outcomes, here I use the preference information contained in a CP-theory to inform how answers to queries over the ontology should be ranked. Finally, in the context of information retrieval, in [17] Wordnet is used to add semantics to CP-net variables.

Another interesting approach to mixing qualitative or quantitative preferences with Semantic Web technology is presented in [83, 149], where an extension of

SPARQL [84] is studied that can encode user preferences in the query.

Skyline for RDF data sources [38, 148] where one just applies min and max on certain values of the tuples.

An abstract and expressive model for defining preferences for Semantic Web Services [70] that can express preferences for favourite values, alternatives, liked and disliked values, qualitative, interval values, min value, max value, closer to a value, scores, Pareto.

Description logic was used to express preferences and the context [160], where the Description Logic concepts were mapped to SQL queries creating a model build on top of a database management system.

A combination of conditional preferences (very different from CP-theories) with DL reasoning for ranking objects is presented in [123]. There, conditional preferences are exploited in the definition of a ranking function that allows to perform a semantic personalised search and ranking over a set of resources annotated via an ontological description. In [115], Datalog$^{\pm}$ is extended with preference management formalisms closely related to those previously studied for relational databases. The paper also considers skyline and k-rank queries, but the allowed preference statements are more general first-order statements, which are not related to CP-theories. This also comes at the price of higher complexity and data tractability results that hold only for disjunctions of atomic queries and not for CQs like here.

Computation of the top-k queries for Semantic web [23, 124, 164] using a ranking criteria and scoring function. Preferences for the Linked data with CP-theories have been studied in [145].

Other combinations of Semantic Web formalisms with preference representation and reasoning include the work by [123], which presents a system to rank-order ontologically annotated objects, using a ranking function based on conditional preferences. [115] focus on preference-based query answering on ontological data by extending Datalog$^{\pm}$ with preference management capabilities, called Pref-Datalog$^{\pm}$.

Preferences in Pref-Datalog$^{\pm}$ have the form of general first-order sentences, and so have a higher complexity. Data tractability results also hold only for disjunctions of atomic queries and not conjunctive queries. [51] use ontological axioms to restrict CP-net outcomes.

Evaluating ranked top-k queries in the context of RDFs (the triple language), OWL 2 Profiles (frame-based languages)and RIF (rule language) was studied in [153].

In AI, preference modelling is more concerned with compact representation and computational issues. In this regard, [14] bridges the gaps between the two streams of preference modelling and suggests that most AI formalisms are fragments of a prototypical preference logic. [154] ceteris paribus semantics and specificity to solve when one might have cycles. Other works are on combining conditional logic and optimistic semantics [104], and handling inconsistencies based on a lexicographic ordering of maximal consistent subsets [10].

8.4 Group Modelling

Many studies address the area of group modelling. Indirectly, it is related to the area of social choice (group decision making, i.e., aiming at the decision that is best for a user given the opinion of individuals), which was studied in mathematics, economics, politics, and sociology [137, 155]. Other areas related to social choice are meta-search [125], collaborative filtering [110], and multi-agent systems [173].

Current approaches that deal with group preferences have also been studied in the area of recommender systems [2, 71, 135], which focus on quantitative preferences.

Social choice theory [69] is also relevant, since it seeks to combine preferences to produce a new preference relation; methods range from those using score-based relations (e.g., approval voting) to ones using more general ones (e.g., ranked pairs). In particular, [141] studies possibility/impossibility results generalising properties such as Arrow's theorem to the case in which incomparable elements exist.

Combining a set of partial ranking into a single ranking of items is known as rank

aggregation [56, 64].

There is a lot of work on preferences aggregations [1, 43, 111]

Another line of work that is relevant to mine is that of modelling group decisions; specifically, the development of both theoretical and practical approaches to solve the problem of choosing a set of elements in such a way that the preferences of a group of individuals are addressed as closely as possible. There are many fields that address this topic, such as mathematics, economics, and sociology [137, 155]. In detail, the recommendation systems for a group of users [2, 110] are closely related to the present work, though they are less general in the sense that queries are not explicitly issued.

The problem of aggregating preferences for a group of users is similar to that of ranking aggregation (or rank fusion), for which there is a quite extensive literature given that several problems in Web applications can be reduced to this problem. In particular, much work has been done on meta-search, i.e., the combination of result lists returned by multiple search engines in response to a given query; in general, these individual results are sorted lists of elements (URLs) accompanied by a relevance score. In [143], several methods for rank aggregation are studied, and experimentally compared for the meta-search problem, while [65] studies an alternative approach to rank aggregation based on decision rules identifying positive and negative reasons for judging whether an element should get a better rank than another (based on two basic principles: majority and respect of minorities). In a related approach from recommender systems, [99] introduces a "blend" operator that combines several recommendations consisting of scores assigned to tuples—this is accomplished via a blending method that is part of the input. Independently of the method used, generally the first step is to normalise the scores assigned to the items over the whole set of rankings, for which there are also several proposals in the literature (cf. [66, 143]).

Chapter 9

Conclusions

This chapter gives a brief overview and summary of the results achieved in this thesis and discuss meaningful conclusions one can draw from them. In addition, it suggests general directions for future research regarding personalisation for ontological query answering.

9.1 Overview

In this thesis I have analysed different preferences languages found in artificial intelligence, and explored which of them are more fit for integration with the Semantic Web. To this end I have concentrated on simple partial qualitative preferences that can handle uncertainty, conditional qualitative preferences (e.g., "If I go to the sea then I prefer wine over beer"), and score based preferences. For each of the resulting languages I have defined their syntax and semantic, what a top-k answer is, algorithm to compute top-k answers. I have compared them formally (e.g., complexity of the algorithms, properties) and empirically with experiments.

I have first presented PP_Datalog$^{\pm}$ an extension of the Datalog$^{\pm}$ for qualitative preference-based query answering under uncertainty - a very expressive language. The focus of this work has been on defining preference combination operators, which produce a preference relation, given a general SPO and a score-based SPO. I have

investigated four specific algorithms for such an operator, and analogised their semantic and computational properties. Finally, I have studied a basic algorithm for answering k-rank queries, and showed that under certain conditions, k-rank queries can be answered in polynomial time in the data complexity, which is the same complexity as answering traditional preference-based queries in relational DBs. I have also reported on an implementation and experimental results.

I have extended PP_Datalog$^{\pm}$ for a group of users since social search become more present over the Web. GPP_Datalog$^{\pm}$ allows for partially ordered preferences of groups of users, and the management of probabilistic uncertainty. This is the first combination of ontology languages with group preferences. I have focused on answering k-rank queries in this context, and studied different approaches to computing answers to queries based on group preferences; the main challenge is to find a principled way, in which to combine the preferences of each individual in the group, while also taking into account the preferences induced by the probabilities obtained from the uncertainty model. I have then studied algorithms to answer k-rank queries for disjunctions of atomic queries under these conditions, and showed that it can be done in polynomial time in the data complexity, as long as query answering can also be done so in the underlying classical ontology. Finally, I have presented a prototype implementation of my framework, including an empirical analysis using real-world data and preference models obtained from real users, showing that my approach is feasible in practice. *GP_Datalog*$^{\pm}$ is a sub-fragment of GPP_Datalog$^{\pm}$ that does not have a probabilistic model, therefore no uncertainty.

I have introduced ontological CP-theories (OCP_theories) (resp., its sub fragment OCP-nets), which are a novel combination of Datalog$^{\pm}$ ontologies with CP-theories (resp., CP-nets). I have defined skyline and k-rank answers for CQs to OCP_theoryand OCP-nets. I have also provided an algorithm for computing such skyline and k-rank answers. Furthermore, I have provided a host of precise complexity (including several data tractability) results for deciding consistency, dominance,

Formalism	Uncertainty	Bipolar Pref	Properties	Queries	Implementations
PP_Datalog$^{\pm}$ [121]	Yes	No	Yes	DAQ	Yes [156]
GP_Datalog$^{\pm}$ [118, 119]	Yes	No	No	DAQ	Yes [156]
GPP_Datalog$^{\pm}$ [118, 120]	Yes	No	Yes	DAQ	Yes [156]
OCP-nets [133, 134]	No	No	No	CQ	No
OCP_theories [50]	No	No	No	CQ	No
SP_Datalog$^{\pm}$	No	Yes	No	UNCQ	Yes [156]
GSP_Datalog$^{\pm}$	No	Yes	Yes	UNCQ	Yes [156]

Figure 9.1: Summary of the ontology languages

and CQ skyline membership for OCP_theory. This language is natural and with tractable conjunctive query answering over some of its fragments.

I have proposed an approach to top-k query answering under user preferences in Datalog$^{\pm}$ ontologies, where the queries are unions of conjunctive queries with safe negation, and the preferences are defined via numerical scores, named SP_Datalog$^{\pm}$. Furthermore, we have explored the generalisation of the above approach to the preferences of a group of users. Finally, I have provided experimental results on the performance and quality of my algorithms. The algorithm has very good computational properties and for the first time we can express negation in the queries.

9.2 Brief Discussion of the Results

In order to draw some meaningful conclusions from results given in the previous chapters, I give a brief interpretation of my results.

In this thesis I made several contributions to the area of personalisation for the Social Semantic Web and advanced the state-of-the-art for personalised ontology-based access to Big Data.

For the first time I have proposed languages that deal with the preferences of a group of users and use them to search for the Social Semantic Web or for ontology-based access to Big Data.

Figure 9.1 gives a brief summary of the results and compares the languages from few perspectives. The columns show if the corresponding ontology can express

uncertainty, bipolar preferences (to express undesirable things or things that are nor preferred), if I have analysed some formal properties of the language, what type of queries analysed (CQ-conjunctive queries, DAQ-disjunction of atomic queries or UCQ-union of conjunction queries with safe negation), and whether I have implemented the framework or not.

9.2.1 Formal Analysis and Expressiveness

One of my goals was to understand which language is good to do personalisation for the Semantic Web. First, I proposed three different families of preferences representation: PP_Datalog$^{\pm}$, OCP_theories and SP_Datalog$^{\pm}$. On the technical side, I analysed the complexity of answering queries, their feasibility in practice. I also provided the properties that should be considered when I compare languages.

The main finding is that there is no such thing as a perfect language, each language has its merits and its weakness in terms of the expressibility and computational properties. Therefore, they should not be in competition but rather complete each other. I cannot compare preferences representation languages in terms of succinctness [46] (if you can translate one language into another in polynomial time) since each language have different ingredients.

9.2.1.1 PP_Datalog$^{\pm}$ and GPP_Datalog$^{\pm}$

PP_Datalog$^{\pm}$ and its generalisation for groups GPP_Datalog$^{\pm}$ are very expressive languages for qualitative preferences in the presence of uncertainty. The drawback of expressiveness comes with its computational complexity for answering skyline queries: even for CQs with guarded ontology its intractable (Σ_2^p [115]). This is the reason why I focused most of the attention on analysing the language for top-k DAQ answers, both for single users and group of users.

The weaken of working with partial preferences is the user understanding— in carrying out my experiments with real users, I found that people do not find partial orderings of elements natural (most questions were variations of “Do I have to fill

in all possible pairs?"). Still, many users do not have a complete ordering of the world. Since users express their preferences partially, I observed that many times they provided just few relations, and I would obtain still too many skyline answers.

In terms of expressiveness these languages are unable to express bipolar preferences in comparison with SP_Datalog$^{\pm}$.

9.2.1.2 OCP_theories

OCP_theories are less expressive than PP_Datalog$^{\pm}$ but under certain conditions I obtained tractable CQs answering, and they allow for a compact encoding of conditional preferences over ground atoms. OCP_theories are more compact than PP_Datalog$^{\pm}$ since using its semantics one can encode more relations with one single preference statement, therefore OCP_theories are more space efficient than PP_Datalog$^{\pm}$.

In terms of expressiveness OCP_theory ontology is unable to express bipolar preferences, uncertainty, and preferences of a group of users- which will be discussed in future work. Still OCP_theories are very good to express context based preferences by using the ontological variables as contexts.

9.2.1.3 SP_Datalog$^{\pm}$ and GSP_Datalog$^{\pm}$

SP_Datalog$^{\pm}$ and its generalisation for groups, GSP_Datalog$^{\pm}$ are score based preference languages that are less expressive than PP_Datalog$^{\pm}$ in terms of preference representations. Still, both of them allow expressing bipolar preferences directly in the query using safe negation.

SP_Datalog$^{\pm}$ and GSP_Datalog$^{\pm}$ have polynomial time data complexity for skyline CQs answering, but one would wonder where does one have numbers from for each atom. One may learn them [68], but it is easier to express explicitly qualitative preferences than scores [53].

In comparison with PP_Datalog$^{\pm}$ and GPP_Datalog$^{\pm}$, the ontology languages SP_Datalog$^{\pm}$ and GSP_Datalog$^{\pm}$ are unable to express uncertainty. In terms of

space efficiency these languages are the least space efficient since one needs to store a score for every ground atom in the query.

9.2.2 Practical Applications

There are many areas for which my work helps to develop future application. First of all, I made all implementations open source[1]. I have contributed to creating benchmark datasets representative for my proposed frameworks. I have gathered real world preferences but as well synthetic generated that could be a starting point for base comparison.

The formal analysis (complexity in time and space, and semantic properties) helped us better understand which language would be better, for which scenario depending on what type of expressiveness one wants to achieve. Moreover, negative results for time complexity for answering skyline queries made us search for fragments of these languages that are still expressive but have more practical applications.

I have used different applications for my experiments and examples: movie scenario, real estate scenario, and going out to eat scenario. This shows the range application my languages could be used for if one wants to use them for personalisation for the ontology-based data [97] access to Big Data.

In terms of using this languages for the recommended systems, one could use the quantitative preferences since they are very efficient in terms of the computational power. In case we would like to integrate the presence of the context, than the conditional preferences are more suitable. In case of the qualitative preferences, if one has information only about partial preferences, than this is an ideal thing.

9.2.3 Group of Users Preferences

I have proposed for the first time extensions of ontology languages with the preferences of a group of users. I have provided different aggregation techniques that were analysed. An interesting question that comes up when implementing these tech-

[1] `https://github.com/personalised-semantic-search`

niques in a real-world scenario is how to decide which aggregation technique to use, and on each chapter dedicated to GPP_Datalog$^{\pm}$ (Chapter 4) and GSP_Datalog$^{\pm}$ (Chapter 7) gives the guidelines based on my experiments.

Although I have proven that I can extend my languages for a group of users, perhaps the most evident limitation is the fact that the only information being leveraged when computing answers to k-rank queries are the preferences of the users; this affords a lightweight query answering framework that requires a relatively minimal amount of input from the user. The weakness, however, is that cases may arise, in which users pose completely opposite preferences leading to a tie which cannot be broken without making an arbitrary choice.

9.3 Directions for Future Research

With the investigation in this thesis, I have laid the groundwork for personalisation for the Social Semantic Web. However, this area is a comparatively young research topic, and many interesting and challenging research problems remain open. The following paragraphs describe issues I am planning to tackle in the future:

Further Enrich Existing Languages. Another interesting topic for future research is to generalise the presented approaches with extra desirable features. For example I could add to PP_Datalog$^{\pm}$ or OCP_theories negative preferences (that is what the user does not like at all). Bipolar preferences have been studied in weighted logic [12], conditional logic [11] and graphical [15]. I could use those ideas to enrich my frameworks. Another example is to add uncertainty to OCP_theories and SP_Datalog$^{\pm}$.

Interesting topics of ongoing and future research include the implementation and experimental evaluation of OCP_theories, as well as the investigation and development of optimised special-case and/or approximation algorithms for OCP_theories, and the exploration of further tractable cases of OCP_theories. Another very interesting work would be on extending OCP_theories for multi-agents (when we

have preferences of a group of users rather than one single user), with the goal of developing new aggregation techniques specially tailored for this formalism.

Approximation Techniques and Sub Fragment. Some complexity results are discouraging to be used for the Big Data. Therefore, a future direction should be on finding other fragments of my ontologies that are expressive enough but with very good computational properties. On the other side w could try to find approximation algorithms, and heuristics that could make the framework even more appealing for the Social Semantic Web.

Develop a hybrid language. My approaches presented in this thesis are either qualitative or quantitative or conditional. A future work would be to have a hybrid preference models that captures different semantics. A challenge would be how to rank the queries when we have both types of preferences, and how to solve the inconsistency of their semantics.

Better Evaluations Methods. For group of users personalisation my experimental evaluation shows that new methods for measuring user satisfaction within a group should be developed, perhaps based on the difference between least and most satisfied users, and incorporated into my framework.

One topic of ongoing and future work is to further experimentally evaluate the similarity of preference aggregations and merging strategies to human judgement, in order to select the best suited ones for search and query answering in the Social Semantic Web;

Interesting topics of ongoing and future research for OCP_theories include: the implementation and experimental evaluation of the presented approach, as well as the investigation and development of optimised special-case and/or approximation algorithms, and the exploration of further tractable cases of OCP_theory.

New Aggregation Techniques. Another interesting vein for future development is deriving explanations along with the answers to queries — in social situations,

it is likely that users' satisfaction with the answer is tied to how it was produced; for instance, if close friends heavily influenced the result the user probably would respond better than if the result was influenced by strangers. Towards this end, I propose to explore the use of provenance models [126] as well as approaches based on argumentation [91,142], among others, in my framework. Finally, another interesting social aspect to work on is that of contemplating "past footprints" of group decisions as part of further information that can be leveraged when answering queries over closely related groups — in this thesis, I assumed a one-shot process where this is not applicable, but this kind of information could be valuable, for instance, in enforcing another form of fairness since individuals favoured in the past can be guaranteed to have less weight in future decisions.

Diverse Answers. As I have seen in the algorithms presented I have had many times elements of non determinism regarding the top-k answers. For example many times we might have more than k skyline answers. There are several directions in which the present work can be extended in the future. One could use ideas from the databases on how to give more diverse answers when this happens (see [112] for a literature review).

There are some cases when there are no answers to the query of the users that satisfy all the users preferences. This can be when the databases is incomplete. One interesting direction is to find which preference to weaken, and how to answer the queries.

Preferences Over Time. Preferences change over time. An interesting direction is to capture preferences change over time (literature on preferences and temporal reasoning [90]) with my proposed formalism.

Another direction in this area is preference revision: when new preferences of the users are added to her old ones, what is the complexity of revising the new preferences for my framework. To express the degree of plausibility of the preferences one might use fuzzy preferences relations [139].

References

[1] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. *SIGMOD*, 29(2):297–306, 2000.

[2] S. Amer-Yahia, S.B. Roy, A. Chawla, G. Das, and C. Yu. Group recommendation: Semantics and efficiency. *VLDB Journal*, 2(1):754–765, 2009.

[3] A. Arvanitis and G. Koutrika. Towards preference-aware relational databases. In *Proc. of ICDE'12*, pages 426–437, 2012.

[4] M. J. Atallah and Y. Qi. Computing all skyline probabilities for uncertain data. In *Proc. of PODS'09*, pages 279–287, 2009.

[5] J.F. Baget, Leclaire, M.L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9):1620–1654, 2011.

[6] J.F. Baget and M.L. Mugnier. Extensions of simple conceptual graphs: The complexity of rules and constraints. *J. Artif. Intell. Res.*, 16:425–465, 2002.

[7] J.F. Baget, M.L. Mugnier, S. Rudolph, and M. Thomazo. Walking the complexity lines for generalized guarded existential rules. In *Proc. of IJCAI'15*, pages 712–717, 2011.

[8] J.F. Baget, M.L. Mugnier, and M. Thomazo. Towards farsighted dependencies for existential rules. In *Proc. of RR'11*, pages 30–45. 2011.

[9] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. of ICALP'81*, volume 115, pages 73–85, 1981.

[10] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency management and prioritized syntax-based entailment. In *Proc. of IJCAI'93*, volume 93, pages 640–645, 1993.

[11] S. Benferhat, D. Dubois, S. Kaci, and H. Prade. Bipolar possibilistic representations. In *Proc. of UAI'02*, pages 45–45. Morgan Kaufmann, 2002.

[12] S. Benferhat, D. Dubois, S. Kaci, and H. Prade. Bipolar representation and fusion of preferences on the possibilistic logic framework. In *Proc. of KR'02*, pages 421–432, 2002.

[13] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Sci. Am.*, 284(5):34–43, 2001.

[14] M. Bienvenu, J. Lang, and N. Wilson. From preference logics to preference languages, and back. In *Proc. of KR'10*, pages 214–224, 2010.

[15] S. Bistarelli, M. S. Pini, F. Rossi, and K. B. Venable. From soft constraints to bipolar preferences: Modelling framework and solving issues. *J. Exp. Theor. Artif. Intell.*, 22(2):135–158, 2010.

[16] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE'01*, pages 421–430, 2001.

[17] F. Boubekeur, M. Boughanem, and L. Tamine-Lechani. Semantic information retrieval based on CP-Nets. In *Proc. of FUZZ-IEEE'07*, pages 1–7, 2007.

[18] C. Boutilier. Toward a logic for qualitative decision theory. In *Proc. of KR'94*, pages 75–86, 1992.

[19] C. Boutilier, F. Bacchus, and R. I Brafman. UCP-networks: A directed graphical representation of conditional utilities. In *Proc. of UAI'01*, pages 56–64, 2001.

[20] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.*, 21:135–191, 2004.

[21] C. Boutilier, R. I Brafman, C. Domshlak, H. H Hoos, and D. Poole. Preference-based constrained optimization with CP-nets. *Comput. Intell.*, 20(2):137–157, 2004.

[22] S. Bouveret, U. Endriss, and J. Lang. Conditional importance networks: A graphical language for representing ordinal, monotonic preferences over sets of goods. In *Proc. of IJCAI'09*, pages 67–72, 2009.

[23] A. Bozzon, E. Della Valle, and Sa. Magliacane. Extending SPARQL algebra to support efficient evaluation of top-k SPARQL queries. In *Search Computing*, pages 143–156. Springer, 2012.

[24] R. Brafman and C. Domshlak. Database preference queries revisited. Technical Report TR2004-1934, Cornell University, 2004.

[25] R. I. Brafman, C. Domshlak, and S.E. Shimony. On graphical modeling of preference and importance. *J. Artif. Intell. Res.*, 25(1):389–424, 2006.

[26] G. Brewka. Preferences, contexts and answer sets. In *Proc. of ICLP'07*, page 22, 2007.

[27] G. Brewka, S. Benferhat, and D. Le Berre. Qualitative choice logic. *Artif. Intell.*, 157(1):203–237, 2004.

[28] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR'08*, pages 70–80, 2008.

[29] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

[30] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *Proc. of PODS'09*, pages 77–86, 2009.

[31] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[32] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *VLDB Journal*, 3(1-2):554–565, 2010.

[33] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in Datalog+/–. In *Proc. of RR'10*, pages 1–17, 2010.

[34] A. Calì, G. Gottlob, and A. Pieris. New expressive languages for ontological query answering. In *Proc. of AAAI'11*, pages 1541–1546, 2011.

[35] A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[36] A. K. Chandra, H.R Lewis, and J.A. Makowsky. Embedded implicational dependencies and their inference problem. *Proc. of STOC'81*, pages 342–354, 1981.

[37] A. K Chandra and P. M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOC'77*, pages 77–90, 1977.

[38] L. Chen, S. Gao, and K. Anyanwu. Proc. of ESWC'11. In *Efficiently Evaluating Skyline Queries on RDF Databases*, pages 123–138, 2011.

[39] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[40] J. Chomicki. Database querying under changing preferences. *Ann. Math. Artif. Intell.*, 50:79–109, 2007.

[41] J. Chomicki. Logical foundations of preference queries. *IEEE Data Eng. Bull.*, 34(2):3–10, 2011.

[42] P. Ciaccia. Querying databases with incomplete CP-nets. In *Proc. of M-PREF'07*, 2007.

[43] V. Conitzer. *Computational aspects of preference aggregation.* PhD thesis, IBM, 2006.

[44] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *Proc. of ICDE'09*, pages 305–316, 2009.

[45] C. Cornelio, A. Loreggia, and V. Saraswat. Logical conditional preference theories. *arXiv preprint arXiv:1504.06374*, 2015.

[46] S. Coste-Marquis, J. Lang, P. Liberatore, and P. Marquis. Expressive power and succinctness of propositional languages for preference representation. In *Proc. of KR'04*, pages 203–212, 2004.

[47] R. da Silva Neves and S. Kaci. Combining totalitarian and ceteris paribus semantics in database preference queries. *Logic Journal of IGPL*, 18(3):464–483, 2010.

[48] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[49] F. D. De Saint-Cyr, J. Lang, and T. Schiex. Penalty logic and its link with Dempster-Shafer theory. In *Proc. of UAI'94*, pages 204–211, 1994.

[50] T. Di Noia, T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Combining existential rules with the power of CP-theories. In *Proc. of IJCAI'15*, pages 2918–2925, 2015.

[51] T. Di Noia, T. Lukasiewicz, and G. I. Simari. Reasoning with semantic-enabled qualitative preferences. In *Proc. of SUM'13*, pages 374–386, 2013.

[52] P. Domingos and W.A. Webb. A tractable first-order probabilistic logic. In *Proc. of AAAI'12*, pages 1902–1909, 2012.

[53] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade. Preferences in AI: An overview. *Artif. Intell.*, 175(7-8):1037–1052, 2011.

[54] C. Domshlak, S. Prestwich, F. Rossi, K. B. Venable, and T. Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *Journal of Heuristics*, 12(4-5):263–285, 2006.

[55] D. Dubois, J. Lang, and H. Prade. Possibilistic logic. *Handbook of Logic in Artificial Intelligence and Logic Programming*, 3:439–513, 1994.

[56] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proc. of WWW'01*, pages 613–622, 2001.

[57] M. Endres and W. Kießling. Transformation of TCP-nets Queries into Preference Database Queries. In *Proc. of ECAI'06*, pages 23–30, 2006.

[58] B. M. Evans and E. H. Chi. Towards a model of understanding social search. In *Proc. of CSCW'08*, pages 485–494, 2008.

[59] R. Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.

[60] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. of ICDT'03*, pages 207–224. 2003.

[61] R. Fagin, P. G Kolaitis, R. J Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[62] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003.

[63] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003. Special Issue on PODS 2001.

[64] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[65] M. Farah and D. Vanderpooten. An outranking approach for rank aggregation in information retrieval. In *Proc. of SIGIR'07*, pages 591–598, 2007.

[66] M. Fernández, D. Vallet, and P. Castells. Probabilistic score normalization for rank aggregation. In *Proc. of ECIR'06*, pages 553–556, 2006.

[67] M. Finger, R. Wassermann, and F. Gagliardi Cozman. Satisfiability in $\mathcal{EL}$ with sets of probabilistic ABoxes. In *Proc. of DL'11*, 2011.

[68] J. Fürnkranz and E. Hüllermeier. *Preference Learning: An Introduction*. Springer, 2011.

[69] W; Gaertner. *A Primer in Social Choice Theory: Revised edition*. Oxford University Press, 2009.

[70] J. M. García, D. Ruiz, and A. Ruiz-Cortés. A model of user preferences for semantic services discovery and ranking. In *The Semantic Web: Research and Applications*, pages 1–14. Springer, 2010.

[71] M. Gartrell, X. Xing, Q. Lv, A. Beach, R. Han, S. Mishra, and K. Seada. Enhancing group recommendation by incorporating social relationship interactions. In *Proc. of GROUP'10*, pages 97–106, 2010.

[72] J. Goldsmith, J. Lang, M. Truszczynski, and N. Wilson. The computational complexity of dominance and consistency in CP-nets. *J. Artif. Intell. Res.*, 33:403–432, 2008.

[73] C. Gonzales, P. Perny, and S. Queiroz. GAI-networks: Optimization, ranking and collective choice in combinatorial domains. *Foundations of computing and decision sciences*, 33(1):3–24, 2008.

[74] G. Gottlob, T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Query answering under probabilistic uncertainty in Datalog+/– ontologies. *Ann. Math. Artif. Intell.*, 69(1):37–72, 2013.

[75] G. Gottlob, T. Lukasiewicz, and G. I. Simari. Answering threshold queries in probabilistic Datalog+/– ontologies. In *Proc. of SUM'11*, pages 401–414, 2011.

[76] G. Gottlob, M. Manna, and A. Pieris. Combining decidability paradigms for existential rules. *Theory and Practice of Logic Programming*, 13(4-5):877–892, 2013.

[77] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE'11*, pages 2–13, 2011.

[78] G. Gottlob, G. Orsi, and A. Pieris. Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014.

[79] G. Gottlob, G. Orsi, A. Pieris, and M. Šimkus. Datalog and its extensions for semantic Web databases. In *Reasoning Web International Summer School*, pages 54–77, 2012.

[80] K. Govindarajan, B. Jayaraman, and S. Mantha. Preference queries in deductive databases. *New Generat. Comput.*, 19(1):57–86, 2001.

[81] K. Govindarajan, B. Jayaraman, S. Mantha, et al. Preference logic programming. In *Proc. of ICLP'95*, pages 731–745, 1995.

[82] S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *VLDB Journal*, 3(1-2):93–104.

[83] M. Gueroussova, A. Polleres, and S. McIlraith. SPARQL with qualitative and quantitative preferences. In *Proc. of OrdRing'13*, pages 2–8, 2013.

[84] S. Harris, A. Seaborne, and E. Prudhommeaux. SPARQL 1.1 query language. *W3C Recommendation*, 21, 2013.

[85] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal*, 13(3):207–221, 2004.

[86] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[87] D. S. Johnson. A catalog of complexity classes. *Handbook of Theoretical Computer Science, volume A*, pages 67–161, 1990.

[88] D.S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28(1):167–189, 1984.

[89] J. C. Jung and C. Lutz. Ontology-based access to probabilistic data with OWL QL. In *Proc. of ISWC'12*, pages 182–197, 2012.

[90] S. Kaci. Preferences aggregation. In *Working with Preferences: Less Is More*, pages 121–150. Springer, 2011.

[91] S. Kaci. Preferences in argumentation theory. In *Working with Preferences: Less Is More*, pages 121–150. Springer, 2011.

[92] S. Kaci. Working with preferences: Less is more. In *Preferences Modeling*. Springer, 2011.

[93] S. Kaci and H. Prade. Relaxing ceteris paribus preferences with partially ordered priorities. In *Proc. of ECSQARU'07*, pages 660–671, 2007.

[94] S. Kaci and L. van der Torre. Reasoning with various kinds of preferences: logic, non-monotonicity, and algorithms. *Annals of Operations Research*, 163(1):89–114, 2008.

[95] W. Kießling. Foundations of preferences in database systems. In *Proc. of VLDB'02*, pages 311–322, 2002.

[96] J. H. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *Proc. of IJCAI'83*, pages 190–193, 1983.

[97] R. Kontchakov, M. Rodríguez-Muro, and M. Zakharyaschev. Ontology-based data access with databases: A short course. In *Proc. of RW'13*, pages 194–229, 2013.

[98] F. Koriche. Relational networks of conditional preferences. *Machine learning*, 89(3):233–255, 2012.

[99] G. Koutrika, B. Bercovitz, and H. Garcia-Molina. FlexRecs: expressing and combining flexible recommendations. In *Proc. of SIGMOD'09*, pages 745–758, 2009.

[100] M. Kronegger, M. Lackner, A. Pfandler, and R. Pichler. A parameterized complexity analysis of generalized CP-nets. In *Proc. of AAAI'14*, pages 1091–1097, 2014.

[101] M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. *Proc. of IJCAI'11*, pages 963–968, 2011.

[102] M. Krötzsch and S. Rudolph. Revisiting acyclicity and guardedness criteria for decidability of existential rules. Technical Report 3011, Institute AIFB, Karlsruhe Institute of Technology, 2011.

[103] M. Lacroix and P. Lavency. Preferences: Putting more knowledge into queries. In *Proc. of VLDB'87*, volume 87, pages 1–4, 1987.

[104] J. Lang. Conditional desires and utilities: An alternative logical approach to qualitative decision theory. In *Proc. of ECAI'96*, pages 318–322, 1996.

[105] N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently computable Datalog programs. In *Proc. of KR'12*, pages 13–23, 2012.

[106] C. Li, K. C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *Proc. of SIGMOD'05*, pages 131–142, 2005.

[107] M. Li, Q. B. Vo, and R. Kowalczyk. An Efficient Majority-Rule-Based Approach for Collective Decision Making with CP-Nets. In *Proc. of KR'10*, 2010.

[108] M. Li, Q. B. Vo, and R. Kowalczyk. Efficient heuristic approach to dominance testing in CP-nets. In *Proc. of AAMAS'11*, pages 353–360, 2011.

[109] X. Lin, Y. Zhang, W. Zhang, and M. A. Cheema. Stochastic skyline operator. In *Proc. of ICDE'11*, pages 721–732, 2011.

[110] G. Linden, B. Smith, and J. York. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[111] F. Liu. *Changing for the better: Preference dynamics and agent diversity.* ILLC, 2008.

[112] C. Lofi and W. Balke. On skyline queries and how to choose from Pareto sets. In *Advanced Query Processing*, pages 15–36. Springer, 2013.

[113] T. Lukasiewicz, M. V. Martinez, G. Orsi, and G. I. Simari. Heuristic ranking in tightly coupled probabilistic description logics. In *Proc. of UAI'12*, pages 554–563, 2012.

[114] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Consistent answers in probabilistic Datalog+/– ontologies. In *Proc. of RR'12*, pages 156–171, 2012.

[115] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Preference-based query answering in Datalog+/-ontologies. In *Proc. of IJCAI'13*, pages 1017–1023, 2013.

[116] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Preference-based query answering in probabilistic Datalog+ ontologies. In *Proc. of ODBASE'13*, pages 501–518, 2013.

[117] T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Group preferences for query answering in Datalog+/– ontologies. In *Proc. of SUM'13*, pages 360–373, 2013.

[118] T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Query answering in Datalog+/– ontologies under group preferences and probabilistic uncertainty. In *Proc. of DMSSW'13*, pages 192–206, 2013.

[119] T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Query answering in probabilistic Datalog+/– ontologies under group preferences. In *Proc. of WI'13*, pages 171–178, 2013.

[120] T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Ontology-based query answering with group preferences. *ACM Transactions on Internet Technology (TOIT)*, 14(4):25:1–25:24, 2014.

[121] T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Preference-based query answering in probabilistic Datalog+ ontologies. *Journal on Data Semantics*, 4(2):81–101, 2015.

[122] T. Lukasiewicz, M.V. Martinez, A. Pieris, and G. I. Simari. From classical to consistent query answering under existential rules. In *Proc. of AAAI'15*, pages 1546–1552, 2015.

[123] T. Lukasiewicz and J. Schellhase. Variable-strength conditional preferences for ranking objects in ontologies. *J. Web Sem.*, 5(3):180–194, 2007.

[124] S. Magliacane, A. Bozzon, and E. Della Valle. Efficient execution of top-k SPARQL queries. In *Proc. of ISWC'12*, pages 344–360, 2012.

[125] M. Manoj and E. Jacob. Information retrieval on internet using meta-search engines: A review. *J. Sci. Ind. Res.*, 67(10):739–746, 2008.

[126] U. Marjit, K. Sharma, and U. Biswas. Provenance representation and storage techniques in linked data: A state-of-the-art survey. *Int. J. Comput. Appl.*, 38(9):23–28, 2012.

[127] J. Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Model. User-Adapt. Interact.*, 14(1):37–85, 2004.

[128] M. Milani and L. Bertossi. Tractable query answering and optimization for extensions of weakly-sticky Datalog+/–. *arXiv:1504.03386*, 2015.

[129] D. Mindolin and J. Chomicki. Hierarchical CP-networks. In *Proc. of M-PREF'07*, 2007.

[130] M. R. Morris. A survey of collaborative Web search practices. In *Proc. of CHI'08*, pages 1657–1660, 2008.

[131] A. Natsev, Y. Chang, J. R. Smith, C. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *Proc. of VLDB'01*, pages 281–290, 2001.

[132] J. Noessner and M. Niepert. ELOG: A probabilistic reasoner for OWL EL. In *Proc. of RR'11*, pages 281–286, 2011.

[133] T. Di Noia, T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Computing k-rank answers with ontological CP-nets. In *Proc. of PRUV'14*, pages 74–87, 2014.

[134] T. Di Noia, T. Lukasiewicz, M.V Martinez., G. I. Simari, and O. Tifrea-Marciuska. Computing k-rank answers with ontological CP-nets. In *Proc. of SEBD'14*, pages 276–283, 2014.

[135] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H.-P. Kriegel. Fast group recommendations by applying user clustering. In *Proc. of ER'12*, pages 126–140. 2012.

[136] C. H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[137] P. K. Pattanaik. *Voting and Collective Choice: Some Aspects of the Theory of Group Decision-Making*. Cambridge University Press, 1971.

[138] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proc. of VLDB'07*, pages 15–26, 2007.

[139] P. Perny and M. Roubens. *Fuzzy Sets in Decision Analysis, Operations Research and Statistics*, chapter Fuzzy Preference Modeling, pages 3–30. Springer, 1998.

[140] G. Pigozzi, A. Tsoukiàs, and P. Viappiani. Preferences in artificial intelligence. *Ann. Math. Artif. Intell.*, 77(3):361–401, 2016.

[141] M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Aggregating partially ordered preferences. *J. Log. Comput.*, 19(3):475–502, 2009.

[142] I. Rahwan and G. R. Simari. *Argumentation in Artificial Intelligence*. Springer, 1st edition, 2009.

[143] M. E. Renda and U. Straccia. Web metasearch: Rank vs. score based rank aggregation methods. In *Proc. of SAC'03*, pages 841–846, 2003.

[144] M. Richardson and P. Domingos. Markov logic networks. *Mach. Learn.*, 62:107–136, 2006.

[145] J. Rosati, T. Di Noia, T. Lukasiewicz, R. De Leone, and A. Maurino. Preference queries with ceteris paribus semantics for linked data. In *Proc. of OTM'15*, pages 423–442, 2015.

[146] G. R. Santhanam, S. Basu, and V. Honavar. Efficient dominance testing for unconditional preferences. In *Proc. of KR'10*, pages 590–592, 2010.

[147] G. R. Santhanam, S. Basu, and V. Honavar. Representing and reasoning with qualitative preferences for compositional systems. *J. Artif. Intell. Res.*, 42:211–274, 2011.

[148] M. Sessoms and K. Anyanwu. Skypackage: From finding items to finding a skyline of packages on the semantic Web. In *Proc. of JIST'12*, pages 49–64, 2013.

[149] W. Siberski, J. Z. Pan, and U. Thaden. Querying the Semantic Web with preferences. In *Proc. of ISWC'06*, pages 612–624, 2006.

[150] M.A. Soliman, I.F. Ilyas, and K. Chen-Chuan Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE'07*, pages 896–905, 2007.

[151] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM T. Database Syst.*, 36(3):19:1–19:45, 2011.

[152] U. Straccia. Top-k retrieval for ontology mediated access to relational databases. *Information Sciences*, 198:1–23, 2012.

[153] U. Straccia. On the top-k retrieval problem forontology-based access to databases. In *Flexible Approaches in Data, Information and Knowledge Management*, pages 95–114. Springer, 2014.

[154] S. W. Tan and J. Pearl. Specification and evaluation of preferences for planning under uncertainty. *Proc. of KR'94*, pages 530–539, 1994.

[155] A. D. Taylor. *Social Choice and the Mathematics of Manipulation*. Cambridge University Press, 2005.

[156] O. Tifrea-Marciuska. Personalised semantic search - source code and datasets, 2015.

[157] O. Tifrea-Marciuska. Personalized search for the Social Semantic Web. *The PhD Workshop of the VLDB'15*, 2015.

[158] R. Torlone and P. Ciaccia. Finding the best when it's a matter of preference. In *Proc. of SEBD'02*, pages 347–360, 2002.

[159] M. B. Twidale, D. M. Nichols, and C. D. Paice. Browsing is a collaborative process. *Inf. Process. Manage.*, 33(6):761–783, 1997.

[160] A. H van Bunningen, L. Feng, and P. MG Apers. A context-aware preference model for database querying in an ambient intelligent environment. In *Proc. of DEXA'06*, pages 33–43. Springer, 2006.

[161] L. Van Der Torre and E. Weydert. Parameters for utilitarian desires in a qualitative decision theory. *Applied Intelligence*, 14(3):285–301, 2001.

[162] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proc. of STOC'82*, pages 137–146, 1982.

[163] J. Von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton university press, 2007.

[164] A. Wagner, T. T. Duc, G. Ladwig, A. Harth, and R. Studer. Top-k linked data query processing. In *Proc. of ESWC'12*, pages 56–71. 2012.

[165] H. Wang, X. Zhou, W. Chen, and P. Ma. Top-k retrieval using conditional preference networks. In *Proc. of CIKM'12*, pages 2075–2079, 2012.

[166] H. S. Warren Jr. A modification of Warshall's algorithm for the transitive closure of binary relations. *Commun. ACM*, 18(4):218–220, 1975.

[167] S. Warshall. A theorem on Boolean matrices. *J. ACM*, 9(1):11–12, 1962.

[168] N. Wilson. Extending CP-nets with stronger conditional preference statements. In *Proc. of AAAI'04*, pages 735–741, 2004.

[169] N. Wilson. An efficient upper approximation for conditional preference. In *Proc. of ECAI'06*, pages 472–476, 2006.

[170] N. Wilson. Efficient inference for expressive comparative preference languages. In *Proc. of IJCAI'09*, pages 961–966, 2009.

[171] N. Wilson. Importance-based semantics of polynomial comparative preference inference. In *Proc. of ECAI'12*, pages 852–857, 2012.

[172] N. Wilson, A.M. George, and B. O'Sullivan. Computation and complexity of preference inference based on hierarchical models. *Proc. of IJCAI'15*, pages 3271–3277, 2015.

[173] M. Wooldridge. *An Introduction to Multiagent Systems*. Wiley, 2009.

[174] F. Yaman and M. desJardins. More-or-less CP-networks. In *Proc. of UAI'07*, pages 434–441, 2007.

[175] Yelp. Yelp Dataset Challenge, 2012.

[176] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and systems*, 1(1):3–28, 1978.

[177] X. Zhang. *Probabilities and Sets in Preference Querying*. PhD thesis, University at Buffalo, State University of New York, 2010.

[178] X. Zhang and J. Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *Proc. of ICDE Workshops'08*, pages 556–563, 2008.

[179] X. Zhang and J. Chomicki. Semantics and evaluation of top-k queries in probabilistic databases. *Distributed and Parallel Databases*, 26:67–126, 2009.

[180] X. Zhang and J. Chomicki. Preference queries over sets. In *Proc. of ICDE'11*, pages 1019–1030, 2011.