Efficient Maximum A-Posteriori Inference in Markov Logic and Application in Description Logics

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Jan Noessner
aus Heidelberg

Mannheim, 2014

Betreuer: Dr. Mathias Niepert, University of Washington, USA

Dekan: Professor Dr. Heinz Jürgen Müller, Universität Mannheim, Deutschland

Referent: Professor Dr. Heiner Stuckenschmidt, Universität Mannheim, Deutschland

Korreferent: Professor Dr. Sebastian Riedel, University College London, England

Tag der mündlichen Prüfung: 19.05.2014

Abstract

The maximum a-posteriori (MAP) query in statistical relational models computes the most probable world given evidence and further knowledge about the domain. It is arguably one of the most important types of computational problems, since it is also used as a subroutine in weight learning algorithms. In this thesis, we discuss an improved inference algorithm and an application for MAP queries. We focus on Markov logic (ML) as statistical relational formalism. Markov logic combines Markov networks with first-order logic by attaching weights to first-order formulas.

For inference, we improve existing work which translates MAP queries to integer linear programs (ILP). The motivation is that existing ILP solvers are very stable and fast and are able to precisely estimate the quality of an intermediate solution. In our work, we focus on improving the translation process such that we result in ILPs having fewer variables and fewer constraints. Our main contribution is the *cutting plane aggregation* (CPA) approach which leverages symmetries in ML networks and parallelizes MAP inference. Additionally, we integrate the cutting plane inference [Rie08] algorithm which significantly reduces the number of groundings by solving multiple smaller ILPs instead of one large ILP. We present the new Markov logic engine ROCKIT which outperforms state-of-the-art engines in standard Markov logic benchmarks.

Afterwards, we apply the MAP query to description logics. Description logics (DL) are knowledge representation formalisms whose expressivity is higher than propositional logic but lower than first-order logic. The most popular DLs have been standardized in the ontology language OWL and are an elementary component in the Semantic Web. We combine Markov logic, which essentially follows the semantic of a log-linear model, with description logics to *log-linear description logics*. In log-linear description logic weights can be attached to any description logic axiom. Furthermore, we introduce a new query type which computes the *most-probable 'coherent' world*. Possible applications of log-linear description logics are mainly located in the area of ontology learning and data integration. With our novel log-linear description logic reasoner ELOG, we experimentally show that more expressivity increases quality and that the solutions of optimal solving strategies have higher quality than the solutions of approximate solving strategies.

Zusammenfassung

Die maximum a-posteriori (MAP) Anfrage in statistischen relationalen Modellen berechnet die wahrscheinlichste Welt, wenn Evidenz und weiteres Wissen über die Domäne gegeben sind. Sie ist nachgewiesener Weise eines der wichtigsten Berechnungsprobleme, da sie auch als Unterprogramm in Gewichtslernalgorithmen benötigt wird. In dieser Dissertation diskutieren wir einen verbesserten Inferenzalgorithmus und eine Anwendung für MAP Anfragen. Markov Logik (ML) wird als statistischer relationaler Formalismus verwendet. Markov Logik kombiniert Markov Netzwerke mit Prädikatenlogik, so dass Gewichte an prädikatenlogische Formeln angehängt werden können.

In Bezug auf Anfrageoptimierung optimieren wir existierende Arbeiten, welche MAP Anfragen zu ganzzahligen linearen Programmen (engl. integer linear programs ILP) übersetzen. Die Motivation hierbei ist, dass existierende ILP Löser sehr stabil und schnell sind und eine genaue Abschätzung der Qualität einer Zwischenlösung ermöglichen. In unserer Arbeit wird der Übersetzungsprozess so verbessert, dass wir ILPs mit weniger Variablen und weniger Nebenbedingungen erhalten. Unser Hauptbeitrag ist der *Cutting Plane Aggregations* (CPA) Ansatz, welcher Symmetrien in ML Netzwerken ausnutzt und MAP Anfragen parallelisiert. Zusätzlich integrieren wir den Cutting Plane Inference [Rie08] Algorithmus, welcher die Anzahl der Nebenbedingungen signifikant reduziert indem statt einem großem ILP mehrere kleinere ILPs gelöst werden. Wir präsentieren eine neue Markov Logik Engine ROCKIT, welche existierende Engins, die dem neuesten Stand der Technik entsprechen, in Qualität und Leistung übertrifft.

Anschließend wenden wir MAP Anfragen auf Beschreibungslogiken an. Beschreibungslogiken (eng. description logics DL) sind Wissensrepräsentationsformalismen, dessen Ausdrucksstärke höher als in Aussagenlogik aber niedriger als in Prädikatenlogik ist. Die bekanntesten DLs wurden in der Ontologiesprache OWL standardisiert und sind elementarer Bestandteil des Semantischen Webs. Wir kombinieren Markov Logik, welche im Wesentlichen die Semantik eines log-linearen Modells hat, mit Beschreibungslogiken zu log-linearer Beschreibungslogiken (engl. *log-linear description logics*). In log-linearen Beschreibungslogiken kann jedem Beschreibungslogikaxiom ein Gewicht hinzugefügt werden. Auerdem führen wir einen neuen Anfragetyp ein, welcher die wahrscheinlichste kohärente Welt (engl. *most-probable coherent world*) berechnet. Mögliche Anwendungen von log-linearen Beschreibungslogiken befinden sich hauptsächlich in den Gebieten des Lernens von Ontologien und der Datenintegration. Mit unserem neuen log-linearen Beschreibungslogikableiter ELOG zeigen wir experimentell, dass mehr Expressivität die Qualität erhöht und dass die Lösung von optimalen Lösungsstrategien eine höhere Qualität hat als approximative Lösungsstrategien.

Contents

List of Figures

List of Tables

Acknowledgement

It is a great feeling to write the acknowledgment of a thesis - *my* thesis. When I started four years ago in Mannheim, this was so far away. In fact, it still felt far away when my Professor Heiner Stuckenschmidt told me to start writing this document one year ago. And now, its finished. I can hardly believe it! But this thesis would never have been possible without the help of so many great people.

First, I want to thank you, Mathias Niepert, for your great supervision especially in the first years. Thank you for sharing your exceptional ideas and for endless, fruitful, and really joyful discussions. Although I tried, I never reached your level of knowledge. You are not only the best supervisor that one could have (and I am not saying this now, because I have to) but also became a friend.

The same happened with Christian Meilicke - we became friends. Especially in the beginning and at the end, you always had a listening ear for me and nice suggestions. Stay as you are!

Furthermore, I thank you, Heiner Stuckenschmidt for the great time working for you. You are probably the best boss one can have. You gave me much freedom to freely choose topics I want to do but always took care that I stayed in the right direction. Thank you for giving me enough time to finish this thesis but thank you also for the final push to get it done.

I thank you, Sebastian Riedel, for being the second referee of my thesis and for the very spontaneous invitation to the hackatron in London over New Years Eve. It was a great time!

I also thank my colleges for the great time and for the many very nice evenings with maybe too much beer discussing relevant and not so relevant things. I hope that we will stay in touch. I thank my friends for just being there although the frequency of meetings decreased in the last time and has to be increased again in the near future. I thank my family who supported me all the way through and who always were eager to understand what I am actually doing all the time. I devote you the 'Informal Motivation' and 'Informal Contribution' sections which might help you to get a rough intuition. I thank my friend Volker Wolf for building my self-confidence and skills in early years. Last but not least, I would like to thank my girlfriend Stefanie Schraeder for her love and inspiration.

Chapter 1

Introduction

One major task of artificial intelligence is to find an abstract representation of real-world problems. Most of these modeling problems require the ability of machines to reason about the modeled knowledge. In the past, researchers have mostly been concerned with deterministic knowledge representation, thereby largely ignoring uncertainty. However, many domains can not be correctly represented by only using deterministic facts. On the other hand, there are many *pure* probabilistic models like Bayesian Networks or Markov Models which are not suitable to represent complex relational structures. Thus, the need arose to combine both worlds.

For a long time, there has not been much cross-fertilization between the knowledge representation and the statistical relational artificial intelligence communities. Recently, more and more interaction has taken place, and languages have been developed, which can incorporate both complex relational structure and uncertainty. These languages are summarized under the umbrella of statistical relational languages. One of the most prominent languages is Markov logic [RD06] which has already been applied in many different research fields although it has only recently been developed. Section 1.1.1 outlines several of its applications and provides an intuitive explanation of Markov logic.

Research on inference in statistical relational models has mainly focused on marginal inference, which is the task to compute a-posteriori probabilities. In contrast, the maximum a-posteriori inference (MAP) (sometimes also referred to as most probable explanation (MPE) inference) has not been studied so extensively. The MAP query computes the most probable world given evidence and knowledge about the domain. It is arguably one of the most important types of computational problems since it is also used as a subroutine in weight learning algorithms [RD06, LD07]. Furthermore, most of the current applications of Markov logic use MAP inference. We refer again to Section 1.1.1 for a verification of this claim and an informal introduction to MAP inference. Thus, this thesis focuses on MAP inference.

In the **first part**, we aim to lower the runtime and thus increase the efficiency of MAP queries in Markov logic. Most state-of-the-art Markov logic solvers incor-

porate approximate algorithms in order to lower the solving times. However, the disadvantage of most of these algorithms is that they are not able to determine if the optimal solution is found or whether they are stuck in a local maximum.

Integer linear programming (ILP) is able to compute a relative gap of every intermediate solution which specifies the worst case difference to the optimal solution. In most ILP solvers, users can specify the minimal gap of the final solution. This includes setting very small gaps and, thus, obtaining (close to) optimal solutions. Furthermore, (mixed) ILP is a very established research field. The existing solvers are heavily used in industry and thus are very stable and fast. We refer to Section 1.1.2 for a more in-depth discussion of these advantages and for an example of the construction of an integer linear program.

Consequently, we transform MAP queries to integer linear programs. In particular, we build upon existing work [Rie08] that already translated MAP queries to integer linear programs. We further optimize the translation process by creating as *compact* integer linear programs as possible. Compact in this case refers to reducing the number of required ILP variables and constraints. Our main contribution in this context is the *cutting plane aggregation* (CPA) approach which detects context-specific symmetries within models. The explicit translation of these symmetries to ILP does not only lead to more compact ILPs, but also enables the solver to detect these symmetries and, thus, apply faster solving strategies. For getting an intuition about the aggregation strategy we forward the reader to Section 1.3.1.

In the **second part**, we satisfy a need which mainly occurs in the research areas of ontology learning and ontology matching. In these areas, scientists develop approaches to compute the *degree of confidence* or *thrust* for description logic axioms. However, these confidence values lack a clearly defined semantics. Furthermore, classical description logics, which enable machines to automatically read and infer new knowledge, are only capable of dealing with deterministic knowledge. Probabilistic extensions of classical description logics are mostly theoretical or require probabilities and not confidence values as input. If we naively incorporate all the learned axioms to the description logic knowledge base, we often result in conflicts. Thus, we require inference mechanisms which repair all conflicts generated by weighted description logic axioms by keeping as much learned information as possible. We refer to Section 1.1.3 for further details of this problem statement and intuitive examples about description logics.

In order to respond to that need, we develop a new uncertain description logic called *log-linear description logics*. This logic extends the classical description logics (DL) to cope with degrees of confidence. The syntax of log-linear DLs allows to attach weights to first-order formulas, and its semantics is based on log-linear models. Furthermore, we introduce a new query type which queries for the *most probable coherent ontology*. Section 1.3.2 gives the reader an informal intuition about log-linear description logics and the novel query.

For computing the most probable coherent ontology, the fast solution techniques developed in Part I are of special importance, as they provide an efficient possibility to solve log-linear description logics queries. Especially, our novel most

probable coherent ontology query can be efficiently solved since symmetries are exploited.

In order to make our theoretical findings usable for other scientists from different domains, this thesis presents two systems RockIt and ELog, which implement the theoretical findings. With these systems, the user can formulate and efficiently solve several concrete problems from different domains.

The remaining part of this introductory chapter is organized as follows: First, we want to take a step back and motivate the content of this thesis on an informal level using illustrative examples in Section 1.1. Afterwards, we formulate the research questions (see Section 1.2), which emerge from the needs formulated in Section 1.1. Then, Section 1.3 describes the two main contributions cutting plane aggregation and log-linear description logics on an informal level. The main goals of Section 1.1 and Section 1.3 is to enable readers from a different domain to glimpse the main motivation and contributions of this thesis and to give domain experts a quick and informal overview. Finally, we give an outline of the thesis and expose our contributions within the sections (see Section 1.4) and briefly mention our main publications for this thesis (see Section 1.5).

1.1 Informal Motivation and Problem Description

In the following we motivate the thesis topic and describe the problems addressed within this thesis in an informal way so that readers from other domains can follow this motivation.

Since both parts deal with maximum a-posteriori (MAP) inference, Section 1.1.1 first gives an informal intuition about this inference type on an illustrative example. Then, this section introduces Markov logic as the statistical relational language used throughout this thesis and give an overview of current application areas and, finally, motivates the necessity of improving the performance of MAP inference by pointing out the importance of MAP inference in Markov logic applications.

We motivate our usage of integer linear programming (ILP) for MAP inference in Part I by work out main advantages compared to traditional solving strategies in Section 1.1.2. This section also provides an informal explanation of the construction of integer linear programs using our previous example.

Finally, Section 1.1.3 exposes the need of an extension of classical description logics that is able to cope with degrees of confidence. This need arose especially from the areas of ontology learning and ontology matching where machine learning algorithms create confidence values for description logic axioms. This need is the motivation of the novel uncertain description logic extension presented in Part II.

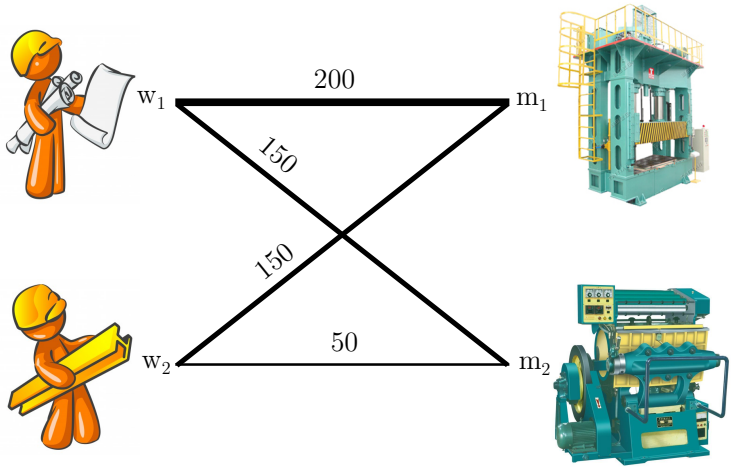

Figure 1.1: Informal example for maximum a-posteriori queries. The example shows workers, who can be assigned to machines. Each assignment generates a profit.

1.1.1 Impact of Maximum A-Posteriori (MAP) Inference in Markov Logic Applications

In maximum a-posteriori (MAP) inference we are interested in computing the most probable world given evidence [KF09]. Let us explain this term with a motivating example, in which we have given two workers w_1 and w_2 and two machines m_1 and m_2. Each worker can work on each machine. However, workers create different amounts of profits when assigned to specific machines. If we assign w_1 to machine m_1 we gain for instance 200. The described world and all existing profit assignments are illustrated in Figure 1.1.

Furthermore, let us assume that there exist two restrictions in our world, saying:

(1) *One worker can be assigned to maximal one machine.*

(2) *One machine can be assigned to maximal one worker.*

These restrictions are called hard, since they must hold in our world and must not be violated. Our goal is now to maximize the sum of the weights of the truth assignments. In this case the total gain of the worker machine assignment can be reinterpreted as probabilities given that we normalize over the sum of all possible outcomes. To that end, this is equivalent to the maximum a-posteriori (MAP) inference query which computes the most probable world.

Let us illustrate on our small example what the sum of the weights of the true assignments means. If we assign worker w_1 to machine m_1, the edge between w_1

and m_1 holds in our world - or in other words the edge is assigned to *true*. Our sum of the truth assignments is 200, because we just have this one assignment.

On a first glimpse, we might also consider this first assignment as a good start towards a most probable world, since we grabbed the highest possible profit first. This behavior is called greedy. According to this greedy strategy, we try to grab the second highest profit next, which is 150. We can realize this profit by either assigning *true* to the pair (w_1, m_2) or to the pair (w_2, m_1). However, both assignments violate our restriction (1) or (2), respectively. Thus, none of these assignments is a *possible* world. Possible worlds are also called models. Last, we can assign worker w_2 to machine m_2, since it does not violate any hard constraints and, consequently, is a model. The sum of the weights of our two truth assignments (w_1, m_1) and (w_2, m_2) is 250.

In our small example, it is easy to see that this is not the assignment with the highest profit. The correct result of the MAP query are the truth assignments (w_1, m_2) and (w_2, m_1), which have a total sum of 300. This small example illustrates that the problem of computing the MAP query is not trivial. A greedy strategy often does not compute the most probable world.

In this thesis, we do not want to restrict our-selves to the proposed worker-machine problem, but want to operate on a formal language. The language we utilize in this thesis is Markov logic [RD06]. An important aspect of this language, which is not covered by our example so far, is the possibility to define so-called soft constraints. Syntactically, soft constraints are constraints with a weight attached. Intuitively, the higher the weight, the higher is the probability that this constraint holds in the final MAP state.

We could for instance turn our second constraint into a soft constraint, such that:

(2) *One machine can be assigned to maximal one worker (weight 100).*

In this case, we get a revenue of 100 for *each* machine if we assign maximal one worker to this machine. Consequently, our state (w_1, m_2) and (w_2, m_1) now has a total sum of the weights of truth assignments of 500. Of this amount, 300 relates to the truth assignments (w_1, m_2) and (w_2, m_1) and $2 \cdot 100$ to the truth assignment of soft rule (2) - one for each machine. If we consider the truth assignments (w_1, m_1) and (w_2, m_1), which were no model before, this combination is now *allowed*. The sum of the weights of these truth assignments is 450. This time only 100 of the 450 are allocated to the soft rule (2) since only machine m_2 gets a 100 reward. Machine m_1 violates the rule because two workers are assigned to it. The MAP state remains (w_1, m_2) and (w_2, m_1) unless we assign a weight *less than* 50 to our soft rule. Then, the MAP state changes to (w_1, m_1) and (w_2, m_1). However, the remarkable difference is that the truth assignments (w_1, m_1) and (w_2, m_1) are now a possible world, since our constraint (2) has become soft.

Due to its declarative nature, Markov logic allows for fast prototyping. Although Markov logic has only been developed in 2006, there already exist numerous applications. One main application area is natural language processing.

More precisely, there exist approaches for improving spoken language understanding [MRRL08a], semantic role labeling [RMR08, MRR09a, MRR09b], weighted abduction [BHD+11], and sentiment analysis [ZNSS11]. Another area is biomedical informatics, where Markov logic has been successfully applied for computing genetic interactions [RK05], predicting the three dimensional structure of proteins [LF09], and extracting bio-molecular events [RCTT09]. Further applications include tasks in the Semantic Web, like ontology matching [NMS10], entity resolution [SD06a, PD08], information extraction [SBS+11], and modeling temporal relations [HBLL10]. Additionally, Markov logic has been applied to other classical machine learning problems like data mining [Dom07] and collective classification [CM11]. Recently, Markov logic has been employed to marketing related topics. Dierkes et al. [DBK11] estimates for example the effect on word of mouth and Singla et. al [SKLG08] detects social relationships in consumer photo collections. Substantially improvements were gained by applying Markov logic in activity recognition [HNS11], where the task is to recognize human activities from sensor data.

Most of the applications of Markov logic incorporate MAP inference. For verification of this claim, we browsed through the first search results for `'Markov Logic'` at Google Schoolar[1] and filtered publications which apply Markov logic. Out of the first 60 search results, 18 publications aims at applications of Markov logic[2]. From these 18 publications, 67% (12 publications) use MAP queries, 22% (4 publications) apply marginal inference, which is the task of computing a-posteriori probabilities, and 11% (2 publications) use both query types. Although this evaluation might not adequately represent all Markov logic applications, it nevertheless shows that the maximum a-posteriori query plays an important role in Markov logic applications.

For these applications, it is crucial to further improve the performance of MAP inference. At the moment, most applications have to restrict the size of their knowledge bases such that it can be handled by state-of-the-art ML solvers. If MAP queries can be solved faster, larger knowledge bases can be processed in less amount of time. Overall, this will further increase the acceptance of Markov logic as representation language. Thus, Part I of this thesis investigates in novel approaches which solve MAP queries faster.

[1] Searched for `http://scholar.google.de/scholar?q=%22Markov+logic%22&btnG=&hl=de&as_sdt=0%2C5` on 25th of March 2014.

[2] The other 42 publications focus on the general presentation of Markov logic in broader contexts or mention ML as general solving technique (12 publications), improve parameter learning (11 publications) or structure learning (8 publications), and extend Markov logic or improve inference mechanisms (11 publications).

1.1.2 Leveraging Integer Linear Programming for Solving Maximum A-Posteriori Queries

Most of the state-of-the-art Markov logic solvers implement approximate approaches for solving MAP queries in Markov logic. Their solving strategy is based on algorithms like MAXWALKSAT which steadily improves the solution by flipping variables. We forward the reader to Section 5.2 for details about the MAXWALKSAT algorithm. Those approximate algorithms might converge to a local maximum without having any possibility to detect that there exist another higher state. Techniques like restarting these algorithms several times with different solutions and randomly flipping variables which do not improve the solution try to overcome local maximas. Nevertheless, these algorithms have no indication about the quality of the current solution. In particular, they have no possibility to provide an estimation on how large the GAP between the actual solution and the optimal solution is.

During the solving process of integer linear programs (ILP) [Sch99] it is always possible to calculate the current lower and upper bound of any solution. The relative distance between the lower and upper bound is referred to as the (relative) gap of a solution. The lower the gap, the higher is the quality of the solution. Thus, ILP solvers allows to specify the relative worst case gap of the actual solution compared to the optimal solution. In fact, most state-of-the-art solvers in integer linear programming allow users to set a relative minimal gap parameter a solution must have. This includes the possibility of setting very small gap parameters and, thus, computing (close-to) optimal solutions. In other words, the user can define the necessary quality of the solution.

Another main advantage of ILP compared to algorithms like MAXWALKSAT is that ILP is not only a very established research field, but its solvers are also heavily applied in industry. The main industrial application area is operations research [WG04]. Operations research includes financial and marketing engineering, transportation, revenue management, and many more business-related areas in which efficient solving strategies for complex real-world problems are required. Around (mixed) integer programming numerous books have been published, like for example [Sch99, Wol00]. Listing scientific work would go beyond the scope of this thesis. Searching for *'Integer Linear Programming'* in Google Scholar returns over 53,000 scientific papers[3]. Thus, there is a large need for very stable and fast solvers like Gurobi [4]. Those solvers implement a very large variety of scientific optimization techniques and are optimized on large real-world benchmarks.

These are the main reasons why integer linear programming has been used in previous work [Rie08] to solve MAP queries in Markov logic networks. In the first part of this thesis, we improve this work by providing novel translation strategies which significantly reduce the size of the ILP translation and leverage symmetries

[3] We searched on the 03.02.2014 for `'Integer Linear Programming'` including quotation marks on `http://scholar.google.de/`.

[4] `http://www.gurobi.com/`

in Markov logic networks.

In principle, integer linear programming maximizes a given *linear* objective subject to *linear* inequality constraints [Sch99]. The variables within the objective and the constraints are *integer*. In order to get an intuition of integer linear programs, we translate our basic problem introduced in Section 1.1.1 to an ILP. We refer the reader to Section 2.4 for a formal introduction of ILPs.

We first have to define ILP variables. In our case, we introduce one variable per *edge* of Figure 1.1. Let $x_{(w_1,m_1)}$ be the variable which represents the edge between worker w_1 and machine m_1. Correspondingly, we define the variables $x_{(w_1,m_2)}$, $x_{(w_2,m_1)}$, and $x_{(w_2,m_2)}$. Next, we set all variables to binary, which means that they can either have the value 1 or the value 0. Semantically, setting the variable to 1 means that the corresponding edge is assigned *true* in our word, and 0 means that the corresponding edge is set to *false*.

Next, we build the objective of our problem. Our aim is to maximize the profit. Thus, our objective is defined as followed

$$obj = \max \;\; 200 \cdot x_{(w_1,m_1)} + 150 \cdot x_{(w_1,m_2)} + 150 \cdot x_{(w_2,m_1)} + 50 \cdot x_{(w_2,m_2)}.$$

If we solve this ILP, every variable x is set to 1 which means that every worker is assigned to every machine. The objective obj of our ILP is 550. However, this violates our restrictions (1) and (2) that we have defined. Thus, we have to add constraints in form of linear inequalities that ensure that our two restrictions are not violated. We add the following four constraints:

(1) *One worker can be assigned to maximal one machine.*

$$x_{(w_1,m_1)} + x_{(w_2,m_1)} <= 1$$

$$x_{(w_1,m_2)} + x_{(w_2,m_2)} <= 1$$

(2) *One machine can be assigned to maximal one worker.*

$$x_{(w_1,m_1)} + x_{(w_1,m_2)} <= 1$$

$$x_{(w_2,m_1)} + x_{(w_2,m_2)} <= 1$$

All of these constraints must hold. The solution of our updated ILP returns $x_{(w_1,m_2)} = x_{(w_2,m_1)} = 1$ and $x_{(w_1,m_1)} = x_{(w_2,m_2)} = 0$ with an objective of 300. This is equivalent to our previously determined MAP state (w_1, m_2) and (w_2, m_1). Again, this example is very informal and should just illustrate the basic ideas of integer linear programs in general and how problems can be translated into its syntax.

In the first part of this thesis, we present several novel translations of Markov logic networks, which include hard and soft formulas, to integer linear programs for solving MAP queries. These improved translations result in more compact ILPs containing fewer variables and fewer constraints than existing translations by exploiting symmetries within Markov logic networks. These techniques decrease the solving times and, consequently, increase the efficiency of MAP inference.

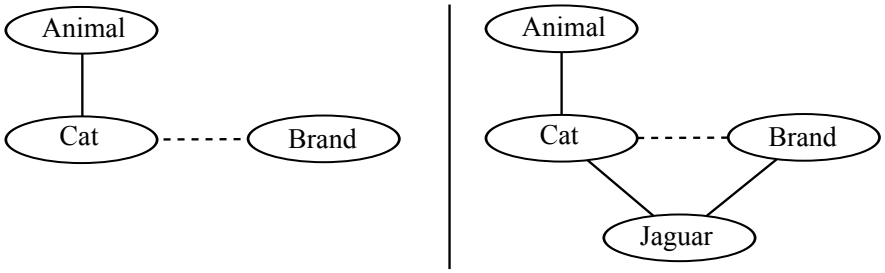

Figure 1.2: Informal example for a coherent (left side) and incoherent (right side) ontology. The circles model classes. The dashed lines symbolize disjointness and the solid lines represent subsumption in which the super-class is located above the sub-class.

1.1.3 Integrating Degrees of Confidence into Description Logics

In real-world applications, uncertainty often occurs as *degree of confidence* or *trust* [NNS11]. The semantic web community, for example, has developed numerous of approaches, which generate confidence values for description logic axioms. Description logics form the theoretical foundations of ontologies. When building an ontology about a certain domain, we enable the machine to not only read the knowledge but to also extract additional knowledge not explicitly stated in the ontology.

Originally originating from philosophy, ontologies nowadays play a central role in the vision of the Semantic Web of Tim Berners-Lee [BLHL$^+$01]. In his vision, intelligent agents can process and manipulate information meaningfully to perform tasks like finding the best hospital for a specific treatment in a radius of 30 miles. To that end, todays web, which is designed for human reading, has to be turned into the Semantic Web, in which information is stored in a machine-readable way. Since ontologies inherit the well-founded semantics of description logics, they are proposed as data representation languages.

The application areas of ontologies are many. Ontologies are frequently used in the medical and bio-informatics domain to structure information [SK02, ABB$^+$00]. Further applications include software engineering [HMS10], libraries [TV03], agriculture [XW07, LLS06], and many more.

One simple example ontology is visualized on the left side of Figure 1.2. In this example, we defined several classes like $Animal$, Cat and $Brand$. Classes are sometimes also called concepts. For simplicity, we omit roles and instances in our example. Figure 1.2 contains two different types of logical dependencies between classes. The first type is subsumption, which is visualized with a solid line, where the super-class is located above the sub-class. We say, for example, that $Animal$ subsumes Cat, or that Cat is a subclass of $Animal$. Semantically,

machines can infer that every Cat is an $Animal$. The second type is disjointness. Disjointness relations are drawn with a dashed line. In our example, we have given that $Animal$ and $Brand$ are disjoint. Thus, nothing can be an $Animal$ and a $Brand$ at the same time. Due to the clearly defined semantics, machines can infer new knowledge like that Cat and $Brand$ are disjoint. This knowledge can not be stated explicitly in the ontology.

After this informal introduction of ontologies, let us take a closer look on the degrees of confidence generated by Semantic Web applications. Two main areas that generate such degrees of confidence for description logic axioms are ontology matching [ES07] and ontology learning [CMSV09, WLB12]. In ontology learning, (semi-)automatic techniques are applied on either structured or unstructured data to learn new axiom types. Several workshops have been taken place in this area [Bre06]. For a general overview of ontology learning approaches we forward the reader to [Zho07, CMSV09, Bre06, WLB12]. In ontology matching we take two ontologies as input and find correspondences between their entities (like concepts or roles). These correspondences often have a degree of confidence attached. Ontology matching is a special case of ontology learning. We forward the reader to Section 10.2.1 and Section 10.2.2 for an explanation and for more in-depth discussions about ontology matching and learning.

An example of such confidence values learned by a tool in ontology learning for subsumption axioms might be the following:

- $Jaguar$ subsumes Cat with confidence value 0.5 and
- $Jaguar$ subsumes $Brand$ with confidence value 1.2.

These confidence values have no clearly defined semantics. Furthermore, classical description logics are no adequate representation of the learned axioms since they often lead to contradictions within the ontology. If we integrate the two learned axioms in our previous example ontology we obtain the ontology depicted on the right side of Figure 1.2. This ontology now contains a contradiction, since $Jaguar$ is defined as a subclass of $Brand$ and Cat but $Brand$ and Cat are inferred to be disjoint. Thus, our ontology states that a $Jaguar$ is a $Brand$ and a Cat although it is forbidden that something is a $Brand$ and a Cat at the same time. This contradiction leads to the conclusion that $Jaguar$ is not allowed to have any elements. In Section 2.5 we formally introduce this contradiction as incoherence. An ontology where no such contradiction exists is called *coherent*.

Hence, formalisms are needed that incorporate these various types of confidence values in a well-defined semantic. Furthermore, new inference types are required which repair all conflicts generated by weighted description logic axioms by keeping as much learned information as possible. In other words, we require novel queries which return the most probable ontology while utilizing the logical concept of coherency.

1.2 Research Questions

After this informal motivation, we formulate precise research questions to address the challanges formulated at the end of Section 1.1.2 and Section 1.1.3. These research questions are split in two parts, which are in line with the general structure of this thesis.

1.2.1 Efficient Inference in Markov Logic

Part I deals with questions on improving the efficiency of maximum a-posteriori queries. In particular, we improve existing techniques which translates the MAP problem into ILP constraints. This includes a novel approach which detects symmetries within Markov logic networks. Leveraging these symmetries leads to more compact ILP translations and, thus, to faster solutions of MAP queries than state-of-the art algorithms.

The research questions which we answer in Part I are as follows:

Q1 *Can we improve existing ILP translation techniques such that we reduce the size of the ILP and make the symmetries of the model more explicit to symmetry detection heuristics?*

Q2 *How can we parallelize the solution process and tightly integrate relational database management systems (RDBMS) within this process?*

Q3 *Does our new techniques reduce runtime and outperform existing Markov logic systems with respect to runtime and quality of the results?*

There are some dependencies between the questions. Since Q1 asks for improvements in existing translations, we first have to clarify how current approaches translate the MAP query to an ILP. Analogously, we first have to state our translation improvements by answering Q1 before we can think about improvements in the solution process in Q2. These improvements include the tight integration of relational database management systems in the solution process and the parallelization of important parts of the this process.

In Q3 we address the experimental verification of the improvements made when answering Q1 and Q2. In exhaustive experiments, we clarify the effects of our improvements on runtime and compare our novel Markov logic engine with other state-of-the art MAP solvers.

1.2.2 Application in Description Logics

Classical description logic is only capable to express deterministic knowledge. In Part II, we discuss a novel possibility of incorporating uncertainty to classical description logics. Other previous approaches have mainly been studied on a theoretical level [Luk08, Cos05, LS08]. Since log-linear models have been integrated in many practical applications [MRRL08a, ZNSS11, NMS10, CM11], we define

a novel uncertain description logic which inherits its semantics from log-linear models. This logic differs from other probabilistic description logics as it allows to attach weights (and not probabilities) to any description logic axiom and as its semantic allows to compute the most probable coherent ontology. Furthermore, we apply extended solving techniques from Part I to efficiently compute the most probable coherent ontology.

Part II answers the following research questions:

Q4 *How can we combine log-linear models with description logics and define the query of a most-probable coherent ontology?*

Q5 *Can we efficiently compute the most-probable coherent ontology utilizing the theory of Part I?*

Q6 *Can we experimentally verify that a solution's quality increase with increasing expressivity and that optimal solving strategies result in higher quality solutions than approximate solving strategies?*

Again, we have dependencies within and between the research questions. We first have to present the general idea of combining log-linear models with description logics and define its exact syntax and semantic which addresses the first part of Q4. From the syntax and semantics, the notion of a most-probable coherent ontology can emerge which copes the second part of Q4. In Q5 we ask if it is possible to apply the solving techniques from Part I for the computation of the most-probable coherent ontology. To that end, we first need to fully understand the notion of a most-probable coherent ontology.

Finally, Q6 aims at the experimental verification of the computation of the most-probable coherent ontology. While both parts of the question ask for an increase in quality, the first part ask for varying expressivity and the second part addresses the optimality of the solving strategy.

1.3 Informal Contribution

This section aims to present the two main contributions of this thesis to readers from other domains in a very informal way utilizing examples. In particular, we intuitively explain our new cutting plane aggregation approach in Section 1.3.1 and our new uncertain description logic called log-linear description logic in Section 1.3.2.

1.3.1 Cutting Plane Aggregation

One of our main contributions in this thesis is a new translation technique to integer linear programs called cutting plane aggregation (CPA) approach [NNS13] which increase the efficiency of MAP queries in Markov logic. The CPA approach improves existing ILP translations such that ILPs have fewer variables and fewer

constraints. Most importantly, the CPA methodology helps the internal symmetry detection techniques of ILP solvers to detect symmetries within the ILP model and thus solve the ILPs faster. We refer the reader to Section 3.4.1 for details about symmetry detection in ILP.

In the following, we give the reader an informal intuition about the basic idea of the aggregation. Please note that the following small example is incomplete since it does not cover soft rules and formulas with disjunction and lacks generalization. Furthermore, we ignore the efficient use of database systems, the integration of an existing efficient optimization called cutting plane inference, and omit parallelization techniques.

For this example, we recapitulate the four ILP constraints for the following restrictions from Section 1.1.2:

(1) *One worker can be assigned to maximal one machine.*

$$x_{(w_1,m_1)} + x_{(w_2,m_1)} <= 1$$

$$x_{(w_1,m_2)} + x_{(w_2,m_2)} <= 1$$

(2) *One machine can be assigned to maximal one worker.*

$$x_{(w_1,m_1)} + x_{(w_1,m_2)} <= 1$$

$$x_{(w_2,m_1)} + x_{(w_2,m_2)} <= 1$$

The cutting plane aggregation algorithm aggregates these four ILP constraints to just two constraints:

$$2x_{(w_1,m_1)} + x_{(w_1,m_2)} + x_{(w_2,m_1)} \leq 2 \qquad (a)$$

$$x_{(w_1,m_2)} + x_{(w_2,m_1)} + 2x_{(w_2,m_2)} \leq 2 \qquad (b)$$

In this example, we halved the number of required constraints from 4 to 2. However, the number of required variables remained constant. Later, we will show that we are also able to reduce the number of required variables significantly when aggregating soft constraints.

Let us go through all of the different cases to verify that the aggregation does not violate restrictions (1) and (2). If we set $x_{(w_1,m_1)} = 1$, it is not allowed to set $x_{(w_1,m_2)} = 1$ or $x_{(w_2,m_1)} = 1$ because this would violate constraint (a). In fact, it is forced due to constraint (a) that $x_{(w_1,m_2)} = x_{(w_2,m_1)} = 0$. Variable $x_{(w_2,m_2)}$ can be set to 1 or 0, but this does not violate restriction (1) or (2). Respectively, if $x_{(w_2,m_2)} = 1$, constraint (b) forces $x_{(w_1,m_2)} = x_{(w_2,m_1)} = 0$.

If $x_{(w_1,m_2)} = 1$ it is not allowed that $x_{(w_1,m_1)} = 1$ because this would violate constraint (a) and it is not allowed to set $x_{(w_2,m_2)} = 1$ due to constraint (b). Thus, we then get $x_{(w_1,m_1)} = x_{(w_0,m_0)} = 0$. Variable $x_{(w_2,m_1)}$ is free to envolve to 0 or 1. However, this does not violate restriction (1) or (2). Correspondingly, if $x_{(w_2,m_1)} = 1$ we also force $x_{(w_1,m_1)} = x_{(w_0,m_0)} = 0$, but allow $x_{(w_2,m_1)} = 0$ or $x_{(w_2,m_1)} = 1$. Again, neither restriction (1) or (2) is violated.

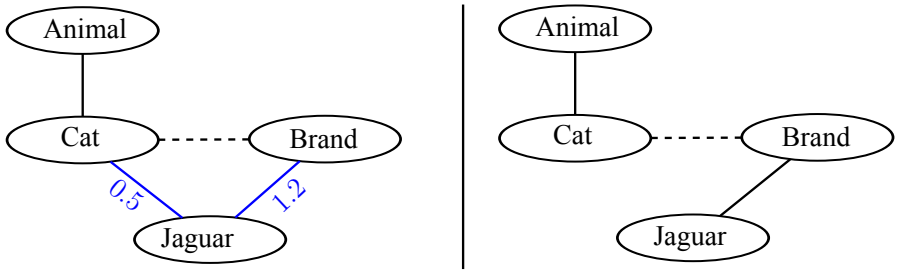

Figure 1.3: Informal example of an ontology in log-linear description logic (left side) and it's corresponding most probable coherent ontology (right side). The circles model classes. The dashed lines symbolize disjointness and the solid lines represent subsumption in which the super-class is located above the sub-class.

1.3.2 Log-Linear Description Logics

Part II introduces a novel uncertain description logic called log-linear description logic [NNS11, NN11]. This logic responds to the need of introducing a well-defined semantics for confidence values attached to description logic axioms formulated in Section 1.1.3. It allows (but do not force) the assignments of weights to any description logic axiom. If we assign no weight, the axiom behaves like an ordinary deterministic axiom. If a weight is assigned, the axiom may or may not hold in our a-posteriori ontology.

In this context we define a novel query type which asks for the most-probable *coherent* ontology. The query produces an ontology that is coherent and entails axioms so that their given a-priori weights are maximized. Intuitively, the higher the weights of the axioms, the more likely they are entailed in the output ontology.

In the following, we give the reader an intuition how log-linear description logics knowledge bases are build and what a most probable coherent ontology is. Again, please notice that the following example is heavily simplified. Exact definitions of syntax and semantics follow in Section 8.2. The example does not cover any solving technique for the most-probable coherent ontology, but rather shows the importance of the problem.

The left side of Figure 1.3 illustrates an example of a log-linear knowledge base where we integrated the confidence values from Section 1.1.3. We have attached weights to some axioms. In particular, we attached the weight 0.5 to Cat subsumes $Jaguar$ and the weight 1.2 to $Brand$ subsumes $Jaguar$.

The most probable *coherent* ontology of our toy example entails $Brand$ subsumes $Jaguar$ but drops the axiom Cat subsumes $Jaguar$ since the weight 1.2 is higher than the weight 0.5. Both axioms can not be in the most probable coherent ontology, because then the ontology would become incoherent. The most probable ontology is visualized on the right side of Figure 1.3.

Please note that with increasing number and higher complexity of the axioms (e.g. by including inter-dependencies between roles and concepts), the computation is far more complex than just taking the maximum value of two numbers. In Part II of this thesis, we provide an efficient solving strategy by transforming the problem into a Markov logic network and applying the solving techniques from Part I to them.

1.4 Outline

This thesis is split in two parts. Part I discusses efficient algorithms for MAP inference in Markov logic. Thereby, our main contribution is the concept of cutting plane aggregation, which optimizes the ILP translation by aggregating hard and soft ground formulas. This aggregation enables symmetry detection algorithms of current ILP solvers to detect symmetries more easily and, thus, compute faster solutions. Part II responds to the need to introduce a well-defined semantic for degrees of confidence by introducing a novel uncertain description logic called log-linear description logic. In particular, we present the idea of computing the most-probable coherent ontology and present efficient ways for computation. This efficient computation apply the improvements of Part I such that we translate log-linear description logics to a Markov logic network and apply our fast solving techniques from Part I.

In the following, we describe the outline of this thesis, give the main references, and highlight our contributions within the respective chapters.

Preliminaries (Chapter 2) In the preliminaries section, we formally introduce the theoretical foundations required for both Part I and Part II. First, we introduce Markov networks mainly utilizing the sources [RD06, Sin12, Kok10, KF09] and first order logic from [GN88]. The maximum a-posteriori (MAP) query is explained in the context of Markov networks. Then, we are in a position to define Markov logic (ML), which is essentially a combination of Markov networks and first order logic, by summarizing mainly [RD06]. Within Markov logic, we formally introduce grounding of Markov logic networks, observed predicates, and types. These concepts are briefly sketched in [Rie08, NRDS11]. However, we explain them in more detail. For example, we explicitly define how evidence reduces the number of groundings.

In Part I we translate Markov logic networks to integer linear programs (ILP). Thus, we formally introduce integer linear programming mainly utilizing [Sch99]. Correspondingly, we define the syntax and semantics of description logics from [BBL05a, NB03], since Part II extends them to log-linear description logics. Some formulations in the preliminary section are taken from our publications [NNS13, NNS11].

1.4.1 Efficient Inference in Markov Logic

MAP Inference in ML with ILP (Chapter 3) Sebastian Riedel [Rie08, Rie09] already applied integer linear programming for solving MAP queries. Section 3.1 presents his translation of the MAP query for Markov logic to ILP. His translation usually requires more than one ILP constraint per ground clause. We then improve his translation in Section 3.2 and provide a novel optimized ILP translation that always requires only one ILP constraint per ground clause, irrespectively if the weight of the ground clause is positive or negative.

After these two translation techniques, Section 3.3 introduces cutting plane inference (CPI) developed by Riedel [Rie08, Rie09] as a meta-algorithm that usually leads to lower solving times. The main idea of cutting plane inference is to add only the violated constraints to the solution and solve several often much smaller ILPs until no more violated constraints remain. The reason for describing CPI is that it can be combined with our novel cutting plane aggregation method as described in Section 3.5.

The cutting plane aggregation (CPA) algorithm is explained in Section 3.4. The CPA method aggregates more than one ground clause to so-called counting constraints. These counting constraints then result in much fewer ILP constraints than without aggregation. Furthermore, the required ILP variables are also reduced. However, most importantly, the aggregation makes symmetries more explicit to state-of-the-art ILP solvers. We managed to parallelize major steps of solving MAP queries. Details are described in Section 3.5. The optimized ILP translation, our novel CPA method, and the parallel MAP algorithm combined with CPI are published in [NNS13].

Leveraging RDBMS (Chapter 4) Inspired by Riedel [Rie08, Rie09], we also apply relational database management systems (RDBMS). We use them for efficient grounding, for finding the violated constraints in each cutting plane inference iteration, and for computing the counting features in our cutting plane aggregation approach. Precise algorithms show how to construct database queries for all three purposes. This chapter contains new and unpublished work which has only been sketched in [NN11, Rie08, Rie09].

Related Work (Chapter 5) Since our cutting plane aggregation method detects symmetries in Markov logic network, we discuss other symmetry detection approaches. Hence, Section 5.1 explains different approaches, which are summarized under the umbrella of lifted inference, from various sources. However, since most of the lifted inference algorithms only cover special cases, no state-of-the art Markov logic system uses symmetry detection methodologies for solving MAP queries. Most frequently, MAP queries in Markov logic are solved with weighted SAT algorithms which are discussed in Section 5.2. They are summarized mainly from [KSJ97, RD06, NRDS11, SD06b]. Finally, Section 5.3 propose details about the existing Markov logic systems against which we compare our algorithms. The

information originates from [Rie08, NRDS11], information on the systems' websites, and personal experience gained while conducting the experiments.

Experiments (Chapter 6) Our experiments compare our novel Markov logic engine ROCKIT against the state-of-the-art Markov logic systems. To that end, Section 6.1 presents ROCKIT and covers some technological details. The experiments are performed on standard Markov logic benchmarks (see Section 6.2). Details about the experimental setup are provided in Section 6.3. Our results in Section 6.4 then empirically show that the cutting plane aggregation method reduces the number of ILP constraints and decreases runtime. Furthermore, our novel MLN engine ROCKIT outperforms other Markov logic systems. Last, our experiments demonstrate that runtime declines with the number of cores. Parts of the experiments have been published in [NNS13].

Conclusion and Future Work (Chapter 7) Finally, we draw a conclusion over Part I and outline future work. Furthermore, Section 7.1 provides a comprehensive example, which shows that the CPA algorithm supports the aggregation of transitive clauses and considers evidence.

1.4.2 Application in Description Logic

Log-Linear Description Logics (Chapter 8) We start Part II with our second main contribution of this thesis: log-linear description logics [NNS11]. Log-linear description logics are a combination of classical description logics with log-linear models. The inference algorithm we propose is restricted to certain description logics, which are summarized from various sources in Section 8.1. For presentation purposes, we choose the description logic $\mathcal{EL}^{++}$ since its complexity is manageable and since it is frequently used to model knowledge in the medical domain. The syntax and semantics are summarized in Section 8.2.

For efficient query solving, log-linear DL knowledge bases are translated to Markov logic networks in Section 8.3. In this context, we also introduce a novel query that returns the most probable *coherent* ontology. This adapted MAP query can be solved efficiently with the solving techniques described in Part I. In general, this chapter is an extended version of our publications [NN11, NNS11] enriched with a presentation of alternative description logics to $\mathcal{EL}^{++}$ (see Section 8.1), the transformation to Markov logic networks (see Section 8.3.5), and several illustrative examples. As we will learn, we require the ability to weight conjunctions of literals. Thus, we have to extend Markov logic such that it can assign weights to any conjunctive normal form. In addition, we provide a novel and not yet published extension of the CPA approach of Part I to conjunctive formulas in Section 8.4.

Related Work (Chapter 9) There have been many attempts to combine classical description logics with some kind of uncertainty. We examine different probabilis-

tic, possibilistic, and fuzzy description logics and compare them to our novel log-linear description logic. Thereby, the main structure is inspired by [LS08], but extended with several other sources. To the best of our knowledge, Section 9.4 lists all existing systems which combine description logics with probabilistic, possibilistic, or fuzzy description logic again aiming at identifying similar systems. Throughout the related work section, we are especially targeting the question whether any other logic subsumes log-linear description logic or can compute some kind of most probable coherent ontology.

Experiments (Chapter 10) Our experiments have two goals. First, we show that increasing expressivity leads to higher quality in the results. Our second focus lies on comparing optimal and approximate approaches showing that optimal approaches have higher quality.

To that end, we first present our novel log-linear description logic reasoner ELOG. Section 10.1 discusses some implementation details. Furthermore, we point out possible application areas of log-linear description logics in ontology learning and ontology matching (see Section 10.2). Thus, our benchmarks introduced in Section 6.2 also originate from these areas. Finally, we describe our experimental setup including different quality measures and discuss experimental results in Section 10.4 and Section 10.5, respectively. In [NNS11] only preliminary experiments in ontology learning have been performed.

Conclusion and Future Work (Chapter 11) Finally, we draw a conclusion and outline future work. This conclusion mainly covers Part II of this thesis. However, it also underlines the interconnection of both parts since Part II exploits the efficient computation methods from Part I.

1.5 Own Published Work

Among all of my publications, I have selected the following three publications as main references for this thesis. All three publications are joint work with my supervisor Dr. Mathias Niepert and two of them are joint work with my supervising professor Prof. Heiner Stuckenschmidt. In this thesis, some text, definitions, lemmas, theorems, figures, and tables are taken from these publications. At the beginning of each Chapter, I precisely specify which sections cover parts of which publication and outline which parts go beyond the respective publication. In order to respect my (co-)authors, I use the more general pronoun 'we' throughout this thesis.

The main publication used in Part I is:

- *Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Rela-*

tional Models. In Proceedings of the 27th Conference on Artificial Intelligence (AAAI), Bellevue, Washington, USA, 2013. [NNS13]

Part II is mainly based on:

- *Mathias Niepert, Jan Noessner, and Heiner Stuckenschmidt. Log-Linear Description Logics. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain, 2011. [NNS11]*
- *Jan Noessner and Mathias Niepert. ELOG: A Probabilistic Reasoner for OWL EL. In Proceedings of the the 5th International Conference on Web Reasoning and Rule Systems (RR), Galway, Ireland, Springer-Verlag, 2011. [NN11]*

Chapter 2

Preliminaries

In the last years, probabilistic graphical models and logics have been combined to statistical relational languages. We refer the reader to [dSBAR05] for a survey over different probabilistic languages. Among all those languages, Markov logic is arguable one of the most popular ones.

In Section 2.3 we define Markov logic as a combination of Markov networks and first order logic. Thus, Section 2.1 and Section 2.2 introduce Markov networks and first order logic, respectively. Marginal inference and maximum a-posteriori (MAP) inference are the two main inference tasks existing in probabilistic graphical models and thus also in statistical relational models. In Section 2.1.2 we define marginal inference, where we aim to find the *real* a-posteriori probabilities of axioms within a model. The goal of the maximum a-posteriori (MAP) inference task, which is further discussed in Section 2.1.2, is to find the most probable world given evidence. In this thesis, we focus on the latter inference task. For inference in Markov networks traditional algorithms often require to ground, that is to instantiate, the whole Markov network. This process is explained in Section 2.3.2.

Since Part I exploits integer linear programming for inference, Section 2.4 formally introduces integer linear programming. Finally, Section 2.5 introduces description logics because Part II extends them to log-linear description logics.

The formal definitions and equations of Section 2.1 are taken from [RD06, Sin12, Kok10] with additional explanations inspired by [KF09] and some other sources. Section 2.1.2 and Section 2.1.3 are a very brief summary of [KF09]. Some text and examples of Section 2.2 and Section 2.3 are taken from our publication [NNS13] extended with further information mainly from [GN88] and [RD06]. Observed predicates and types (see Section 2.3.1) as well as the grounding of Markov logic networks are very briefly sketched in [Rie08, NRDS11]. We explain these concepts in more detail. Furthermore, we explicitly explain how the number of groundings can be reduced due to evidence. The explanation of integer linear programming in Section 2.4 is summarized from [Sch99] extended with several other sources. The main references for the introduction to description logics in Section 2.5 are [BBL05a, NB03, NNS11].

2.1 Markov Networks

Historically, the concept of Markov networks (also known as Markov random fields) has first been discovered in the thesis of the German physicist Ernst Ising [Isi25]. Later, Preston [Pre74] and Spitzer [Spi71] made Markov models available for non-mathematicians and presented applications in other areas [KS80].

Markov networks (also known as Markov random fields) belong to graphical models. A graphical model is a probabilistic model where the conditional dependence between random variables is modeled with a graph. In case of Markov networks, this graph is undirected.

Formally, a Markov network is an undirected graphical model that defines a joint distribution over a set of variables $X = \{X_1, X_2, ..., X_n\} \in \mathcal{X}$ [Pea89]. It is composed of an undirected graph G and a set of potential functions ϕ_k. The values associated with the potential function may be any real number. They do not represent probabilities and thus are not necessary normalized. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A clique is defined as a subset of nodes in a graph such that these nodes are fully connected [BN07]. A potential function is a non-negative real-valued function of the state of the corresponding clique. The state of the clique is composed of all possible states of its nodes. In this thesis, we assume that the variables associated to the nodes are binary ($X_i = true$ or $X_i = false$). For a clique including for instance three nodes, we get $2^3 = 8$ possible states where each state is associated with a real number. The joint distribution represented by a Markov network is given by

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}})$$

where $x_{\{k\}}$ is the state of the variables that occur in the kth clique. Due to the assumed independence between the variables, we compute their probability by multiplying the states of the cliques. Which states we have to multiply is given by the specific truth assignment $X = x$. The partition function Z is defined as the sum over all joint distributions of x

$$Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$$

and ensures that the joint distribution is correctly normalized.

2.1.1 Log-linear Models

There are many different possible representations of the parametrization of Markov networks. Despite the classical representation as a product over potential on cliques, we can also represent them as log-linear models. Log-linear models allow us to incorporate both uncertain and deterministic dependencies between description logic axioms. The ability to integrate heterogeneous features make log-linear models a

commonly used parametrization in areas such as natural language processing and bioinformatics and a number of sophisticated algorithms for inference and parameter learning have been developed. In this thesis, we focus on the representation as log-linear models

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(x)\right)$$

in which each potential is replaced by an exponentiated weighted sum. This thesis focuses on binary features $f_i(x) \in \{0, 1\}$. Here, we have one feature corresponding to each possible state $x_{\{k\}}$ of each clique, with its weight being $\log \phi_k(x_{\{k\}}$. If we have, for example, a clique including three nodes, we get $2^3 = 8$ different features $f_i(x)$. For one particular truth assignment, one out of those eight features is one ($f_i(x) = 1$) and the others are zero. Example 1 illustrates the connection between the classical representation and the representation as log-linear models.

Example 1. *Consider a very simple Markov network containing three binary variables $\{A, B, C\} \in \mathcal{X}$. Let us assume that ($A$ and B) and (B and C) are connected. Thus, our network has two maximal cliques; $k_1 \in \{A, B\}$ and $k_2 \in \{B, C\}$. Furthermore, we assume the following values for the potential functions are given*

ϕ_{k_1}	b^0	b^1
a^0	*1*	*4*
a^1	*2*	*3*

ϕ_{k_2}	c^0	c^1
b^0	*3*	*10*
b^1	*1*	*2*

where e.g. a^1 means A = true and a^0 means A = false respectively. For $X = \{A, B, C\}$ and $x = \{a^1, b^0, c^1\}$ we get the following normal probability distribution

$$P(X = x) = \frac{1}{Z}\phi_{k_1}(a^1, b^0) \cdot \phi_{k_1}(b^0, c^1) = \frac{1}{Z}2 \cdot 10 = \frac{1}{Z}20$$

and the following log-linear distribution

$$\begin{aligned} P(X = x) &= \tfrac{1}{Z} \exp\left(w_{a^1,b^0} f_{a^1,b^0}(x) + w_{b^0,c^1} f_{b^0,c^1}(x)\right) \\ &= \tfrac{1}{Z} \exp\left(\log(10) \cdot 1 + \log(2) \cdot 1\right) = \tfrac{1}{Z}20 \end{aligned}$$

where all other $f_j(x) = 0$. Since Z has not changed, this example shows that the two distributions are equivalent.

We now turn our attention to inference. In Markov networks as well as in all probabilistic graphical models, there exist two main types of inference queries, which are the conditional probability query and the maximum a-posteriori query.

2.1.2 Marginal Inference

The (conditional) probability query computes the posterior probability distribution

$$P(\mathbf{X}|\mathbf{E} = \mathbf{e})$$

over the values $\mathbf{x}$ of (a subset of) all variables $\mathbf{X}$ given the evidence values $\mathbf{e}$ of $\mathbf{E}$. This inference type is also referred to as marginal inference.

Exact inference algorithms can be roughly divided in two classes. The first class consists of algorithms based on variable elimination. Thereby, the exponential blowup is avoided by caching intermediate results and performing the summation of the joint distribution from the inside out rather than from the outside in [KF09, BB72]. The second class of algorithms are based on the sum-product belief propagation, which reformulates the idea of variable elimination. They build a so-called clique tree which edges deliver messages. These messages are equivalent to the factors of the variable elimination algorithm. The nodes of the clique tree are the variables involved in the direct input and output factors. Each clique takes the incoming messages, multiplies them, sums out one or more variables, and sends an outgoing message to another clique. The computation of the posterior probability over all variables can be efficiently implemented using bidirectional messages [SS90, BB72].

Exact marginal inference in Markov Networks is #P-COMPLETE [Rot96]. Thus, a variety of approximative inference algorithms have been developed. One group of algorithms are propagation-based algorithms, which use other data structures like cluster graphs rather than clique trees. Most of these algorithms are based on the so-called loopy belief propagation [Pea89], because the cluster graphs often contain undirected cycles. The idea of this algorithm is that the clusters within the cluster graph are smaller than those in the clique tree. Thus, the message passing steps are less expensive. When propagating the messages the algorithm ignores the existence of loops and permits the nodes to communicate as if there were no loops. After several iterations, the algorithm usually converges to an approximative probability distribution.

Another group consists of algorithms that sample particles from some distribution. A particle is an instantiation to some or all variables. Markov chain Monte Carlo methods [GRS96] or more precisely the Gibbs sampling algorithm [LLR81] is the most used algorithm within this group. The basic idea of Gibbs sampling is the creation of a sequence of samples that get closer and closer to the desired posterior distribution. Instead of always sampling from the a-priori distribution, we fetch only initially a sample based on this distribution. This initial sample is then modified by iterating over each unobserved variables, sampling a new value given the current sample. This new sample can be computed very efficiently, since the assignment of every other value is given.

2.1.3 Maximum A-Posteriori (MAP) Inference

In this thesis, we concentrate on the second type of query, the maximum a-posteriori (MAP) query. Sometimes this type of query is also called most probable explanation. In MAP queries the task is to find the most probable world given evidence. Let $\mathbf{E} = \mathbf{e}$ be the given evidence. Then, we aim to find an assignment of all non-

evidence variables $\mathbf{X} = \mathbf{x}$ so that

$$\underset{\mathbf{x}}{\operatorname{argmax}}\ P(\mathbf{X} = \mathbf{x} \mid \mathbf{E} = \mathbf{e}).$$

In the following, we often call evidence variables observed variables and non-evidence variables hidden variables. The assignment $\mathbf{x}$ which lead to the maximal P is called maximum a-posteriori (MAP) state. If more than one assignment leads to the same result, we can pick a random one. Finding such an assignment is not a trivial problem, since the assignment in which a single potential function picks its most likely value is often not the optimal assignment for the global Markov network. Example 2 illustrates this problem.

Example 2. *Let us recall the Markov network from Example 1. Let us furthermore assume that no evidence is given. If we first take the optimal assignment $\phi_{k_1}(x_1) = 4$ with $x_1 = \{a^0, b^1\}$ for the single potential function ϕ_{k_1}, we have limited choices for the second function ϕ_{k_2}. In fact, the maximal value for ϕ_{k_2} given our previous decision then is $\phi_{k_2}(x_2) = 2$ with $x_2 = \{b^1, c^1\}$. This results in $P(X = x) = \frac{1}{Z}8$ where $x = \{a^0, b^1, c^1\}$. This is not the maximal possible result for P, since we received a higher value in Example 1. In fact, the assignment $x^* = \{a^1, b^0, c^1\}$ from Example 1 is the MAP-state of this small example.*

MAP Inference is NP-COMPLETE [Shi94]. Many algorithms divide MAP-query inference into two parts. First, they compute the maximal unnormalized possible value for the distribution P (also called max-marginals) and then, they extract the corresponding maximal assignment of every variable. For computing the max-marginals we can reuse the variable elimination and sum-product belief propagation algorithms from marginal inference, except that we compute the maximum instead of the sum inside the algorithm. In case of the latter one, we result in higher complexity because the required max-sum operations do not commute. In a second step, we have to extract the corresponding maximal assignments. We cannot determine the maximizing value (e.g. true or false) of a variable while we eliminate variables because we do not yet know which is the optimal strategy during elimination. However, we can determine a maximizing value *given* the values of the variables that have not yet been eliminated. Thus, we can trace back the solution and successively pick the values of the variables until we reached the last eliminated variable. The result then is the most likely assignment.

Many approximate algorithms for marginal inference can be adapted for MAP inference [WF01]. In case of the loopy belief propagation algorithm, we can compute the max-marginals by slightly adapting the message passing algorithm. However, in practice, the algorithm tends to converge less often as, for example, the sum-product algorithm. Decoding the (approximate) max-marginals to the corresponding (approximate) maximal assignments is also much more difficult in loopy cluster graphs. However, it has been shown that the approximate MAP solution is robust against large perturbations and is guaranteed to be a strong local maxima. More recently, Kolmogorov [Kol06] developed a convergent algorithm called TRW-S which guaranteed to converge, although it can get stuck in local maxima.

Approximate algorithms based on sampling are widely used for solving MAP queries. One of the most successful algorithms is discussed in dept in the scope of Markov logic in Section 5.2. Since Markov logic is build on Markov networks (see Section 2.3), the algorithm can also be applied to Markov networks.

2.2 Function Free First Order Logic (FOL)

Since Markov logic combines first order logic and Markov models, we recall some basic concepts of function free first-order logic (FOL) [GN88]. Compared to natural language, the syntax and semantic of first order logic is well-defined and thus given a first-order logic theory we can *mechanically* infer new knowledge. Due to disambiguate formulations and huge parsing and interpretation difficulties this is currently almost impossible for natural language. However, first order logic is limited to hard facts and can not express uncertainty.

In the following, we provide a formal definition of function free first-order logic. A term is either a constant or a variable. An atom $p(t_1, ..., t_n)$ consists of a predicate p/n of arity n followed by n terms t_i. A literal ℓ is an atom a or its negation $\neg a$. We call the former non-negated and the latter negated literal.

Formulas are recursively constructed from literals using logical connectives and quantifiers. If F_1 and F_2 are formulas, the following expressions are also formulas:

- conjunction of formulas ($F_1 \wedge F_2$), which is true iff both F_1 *and* F_2 are true,
- disjunction of formulas ($F_1 \vee F_2$), which is true iff F_1 *or* F_2 is true,
- implication ($F_1 \Rightarrow F_2 \equiv \neg F_1 \wedge F_2$), which is true except if F_1 is true and F_2 is false,
- equivalence ($F_1 \Leftrightarrow F_2 \equiv (F_1 \Rightarrow F_2) \wedge (F_2 \Rightarrow F_1)$, which is true iff F_1 and F_2 have the same truth value, and
- universal quantification ($\forall\ x\ F_1$), which is true iff F_1 is true for every object x in the domain.

In this thesis, we assume for the sake of readability that all formulas are universally quantified. Due to this assumption, formulas do not contain existential quantification ($\exists\ x\ F_1$, which is true iff F_1 is true for one object x in the domain). All presented algorithms and optimizations can be adapted to work for existential quantification.

A clause

$$c = \ell_1 \vee \ldots \vee \ell_k$$

is a disjunction of literals. Horn clauses are clauses with at most one positive literal. Each first order formula can be converted to the prenex conjunctive normal form (CNF) where we first specify all quantifiers and the corresponding variables followed by a conjunction of clauses. Since we omit existential quantification, we get a conjunction of clauses where all variables are assumed to be universally quantified of the form

$$(\ell_{1,1} \vee \ldots \vee \ell_{1,m}) \wedge \ldots \wedge (\ell_{n,1} \vee \ell_{n,m}).$$

A substitution $\varphi = \{v_1/t_1, ..., v_l/t_g\}$ is an assignment of terms t_i to variables v_i. A theory is a finite set of clauses implicitly defining a conjunction of these clauses. An expression is an atom, literal, clause or theory. An expression is ground if it does not contain any variables. An analogous definition holds for ground atoms, ground literals, and ground clauses. A tautology is a clause which is true in every possible interpretation.

The Herbrand base $\mathcal{H}$ is the set of all possible ground atoms. A Herbrand interpretation is a subset of the Herbrand base. All ground atoms in the Herbrand interpretation are assumed to be true while all others are assumed to be false. A Herbrand interpretation I is a Herbrand model of a set of first-order formulas $\mathcal{S}$, written as $\models_I \mathcal{S}$, if and only if it satisfies all groundings of formulas in $\mathcal{S}$. A Herbrand interpretation I satisfies a clause c, written as $I \models c$, if there exists a ground atom ℓ (negated ground atom $\neg\ell$) in c with $\ell \in I$ ($\ell \notin I$). A Herbrand interpretation I satisfies a theory T, written as $I \models T$, if it satisfies all the clauses $c \in T$.

2.3 Markov Logic (ML)

Markov logic [RD06] is a first-order template language for undirected graphical models like Markov networks. It combines first-order logic with log-linear models by assigning weights to first-order formulas. The formalism allows one to build knowledge bases incorporating both, deterministic and probabilistic knowledge.

Due to its expressiveness and declarative nature, numerous real-world problems have been modeled with Markov logic. Especially in the realm of knowledge management applications such as data integration [WW08, NMS10, SD06a], ontology refinement [WW08], and information extraction [PD07, KD08], Markov logic allows for rapid prototyping and competitive results. Other examples include spoken language understanding [MRRL08b], event modeling [TD08], and activity recognition [HNS11] to name but a few.

More formally, a Markov logic network $\mathcal{M}$ is a finite set of pairs $(F_i, w_i), 1 \leq i \leq n$, where F_i is a clause in function-free first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, ..., c_n\}$, it defines a Markov network, the ground Markov logic network $\mathcal{M}_C$, with

- one binary node for each possible grounding of each predicate occurring in $\mathcal{M}$ and
- one feature for each possible grounding of each clause F_i occurring in $\mathcal{M}$ with feature weight w_i.

Hence, a Markov logic network defines a log-linear probability distribution over Herbrand interpretations (possible worlds). For a possible world $\mathbf{x}$ the probability is defined as

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(\mathbf{x})\right).$$

weight	clause
1.4	$\neg\mathsf{smokes}(x)$
2.3	$\neg\mathsf{cancer}(x)$
4.6	$\neg\mathsf{friends}(x, y)$
1.5	$\neg\mathsf{smokes}(x) \vee \mathsf{cancer}(x)$
1.1	$\neg\mathsf{smokes}(x) \vee \neg\mathsf{friends}(x, y) \vee \mathsf{smokes}(y)$

Table 2.1: The friends & smokers MLN [SD08].

Herbrand base
$\mathsf{smokes}(\mathsf{Anna}), \mathsf{smokes}(\mathsf{Bob}),$
$\mathsf{cancer}(\mathsf{Anna}), \mathsf{cancer}(\mathsf{Bob}),$
$\mathsf{friends}(\mathsf{Anna}, \mathsf{Anna}), \mathsf{friends}(\mathsf{Anna}, \mathsf{Bob}),$
$\mathsf{friends}(\mathsf{Bob}, \mathsf{Anna}), \mathsf{friends}(\mathsf{Bob}, \mathsf{Bob})$

Table 2.2: The Herbrand base of the friends & smokers MLN [SD08] for two constants Anna and Bob.

where $n_i(\mathbf{x})$ is the number of true groundings of clause F_i in $\mathbf{x}$ and Z is a normalization constant. Every subset of the Herbrand base is an interpretation (possible world). Each ground atom in the Herbrand base corresponds to one binary random variable in the ground Markov logic network.

If $w_i = \infty$ we refer to the formula as hard formula which must hold in the KB, while for $w_i \in \mathbb{R}$ we call the formula a soft formula which holds to a certain degree.

As in Markov networks, we distinguish between MAP inference and marginal inference. We refer the reader to Section 2.1.3 and Section 2.1.2 for details. In this thesis we focus on MAP inference as learning algorithms typically employ inference algorithms as subroutines. Hence, being able to compute MAP states of Markov logic networks more efficiently also helps to improve learning algorithms [RD06, LD07]. In order to compute the MAP state (sometimes also referred to as the maximum probable explanation), given evidence $\mathbf{E} = \mathbf{e}$, we have to compute the maximization problem

$$\operatorname*{argmax}_{\mathbf{x}} P(\mathbf{X} = \mathbf{x} \mid \mathbf{E} = \mathbf{e})$$

where the maximization is performed over all possible worlds (Herbrand interpretations) $\mathbf{x}$ compatible with the evidence. Intuitively, the higher the weight of a clause, the less probable is a possible world violating groundings of said clause within a MAP state.

Example 3. *Table 2.1 depicts the weighted clauses of the Friends & Smokers MLN. The Herbrand base for two constants* Anna *and* Bob *is depicted in Table 2.2.*

2.3.1 Observed Predicates and Types

Inference approaches for Markov logic networks often require a complete grounding which can become intractable large. As prerequisite for the next section, in which we explain the grounding procedure in detail, we introduce two practical ways which reduce the number of groundings and are reused in the following section. The first is observed predicates which are predicates whose groundings are given due to evidence. Second, types *typify* variables so that their constants belong to different scopes.

We start with introducing observed predicates [Rie08]. For observed predicates the closed world assumption holds. This means that all ground atoms of this predicate not listed in the evidence are false or, in other words, each possible grounding of an observed predicate must be given due to evidence $\mathbf{E}$. Thus, we can exclude all groundings of observed predicates from the set of variables $\mathbf{X}$ for which we compute the MAP-state. For distinguishing observed predicates from the *normal* predicates, the latter ones are also called hidden predicates or query predicates. Whenever we just talk about predicates we refer to hidden predicates. Please note that we may define evidence for a *strict* subset of the literals of hidden predicates.

Types define different scopes for variables. These scopes are disjoint set of constants. Formally, a type $\mathcal{T}$ consists of a finite set of constants and is a subset of the Herbrand universe. Example 4 illustrates the usage of types. If we do not explicitly define types (like in Example 3), we assume that we have one global set of constants (Herbrand base) for all variables.

Example 4. *In Example 3 every variable refers to the same scope, containing the constants $Anna$ and Bob. However, if we have a formula like $drives(x, y) \wedge fastCar(y) \Rightarrow fastDriver(x)$ then it is useful to define different types of variable x and variable y. Reasonable type assignments in this example are $x \in \mathcal{P}$ and $y \in \mathcal{C}$, where $\mathcal{P}$ and $\mathcal{C}$ are disjoint sets of constants representing persons and cars, respectively. If we now define the constants $\{Anna, Bob\} \equiv \mathcal{P}$ and $\{Bmw, Audi\} \equiv \mathcal{C}$, we result in 4 ground clauses. Without the definition of types, we obtain 16 ground clauses.*

2.3.2 Grounding

Grounding Markov logic networks is a pre-processing step required for (most) solving techniques. In this section, we construct a set $\mathcal{G}$ containing all ground clauses of a Markov logic network. This set of ground clauses is required as input for the methods presented in Chapter 3.

Similar to constraint propagation techniques [Mac77, Apt99], in which the range of the variables are successively reduced through the evaluation of constraints, we simplify the ground clauses due to the given evidence. Evidence allows us to omit literals or even whole ground clauses.

We begin with constructing the set $\mathcal{G}^{\mathsf{all}}$ which contains all the ground clauses and their weights ignoring evidence. We define a MLN consisting of a finite num-

ber of clauses c with weight w. Each clause consists of a set of variables $\mathcal{X}(c)$. Each variable $x \in \mathcal{X}(c)$ is assigned to one type $\mathcal{T}$. As explained in the last section, a type $\mathcal{T}$ is a subset of the Herbrand universe. The type $\mathcal{T}$ which refers to variable x is denoted with $\mathcal{T}(x)$. Usually, a MLN has only few types shared over multiple clauses and variables. If no type has been assigned to a variable, we assign her a new type. This new type then consists of all constants in the Herbrand universe. The set $\mathcal{G}^{\mathsf{all}}$ is now constructed as followed:

1. Set $\mathcal{G}_{\mathsf{all}}$ to the empty set.
2. For each first-order logic clause c substitute every variable x with every constant $t \in \mathcal{T}(x)$.
3. Add every (variable-free) ground clause g resulting from Step 2 with its weight w to $\mathcal{G}^{\mathsf{all}}$

Note that the number of ground clauses $g \in \mathcal{G}^{\mathsf{all}}$ is usually much larger than the number of clauses c containing variables. For one clause c we result in

$$\prod_{x \in \mathcal{X}(c)} |\mathcal{T}(x)|$$

ground clauses g.

If we consider evidence we can reduce the number and the complexity of ground clauses in $\mathcal{G}^{\mathsf{all}}$. Let $\mathbf{E} = \mathbf{e}$ be the given evidence. For the ground clause $g \in \mathcal{G}^{\mathsf{all}}$ let $L^+(g)$ be the set of ground atoms occurring unnegated in g and $L^-(g)$ be the set of ground atoms occurring negated in g. Let $\mathcal{G}$ be an initially empty set of all ground clauses after considering evidence. For every ground clause $g \in \mathcal{G}^{\mathsf{all}}$ we perform the following:
If any *positive* ground atoms $\ell \in L^+(g)$ in the ground clause g is

- set to *false* by the given evidence, we replace this literal ℓ in g with *false* which is tantamount of deleting ℓ from g or
- set to *true* by the given evidence, we can drop the entire clause g, since it always evolves to *true* (tautology).

In case any *negative* ground atoms $\ell \in L^-(g)$ is

- set to *true* by the given evidence, we can drop out this atom ℓ from the clause g or
- set to *false* by the given evidence, we can drop the entire clause g, since it always evolves to *true* (tautology).

If clause g has not been dropped and if clause g still contains at least one literal, we add g to $\mathcal{G}$.

Intuitively, some ground clauses can be ignored because they evolve to a tautology due to evidence. For other ground clauses we can at least reduce the complexity by omitting some literals due to evidence. Overall, evidence leads to a reduction of the number and the complexity of ground clauses.

Please note that the final set $\mathcal{G}$ only contains hidden literals. These hidden literals consist of hidden predicates only. All literals constructed from observed

weight	clause
1.4	¬smokes(Anna)
1.4	¬smokes(Bob)
2.3	¬cancer(Anna)
2.3	¬cancer(Bob)
4.6	¬friends(Anna, Anna)
4.6	¬friends(Anna, Bob)
4.6	¬friends(Bob, Anna)
4.6	¬friends(Bob, Bob)
1.5	¬smokes(Anna) ∨cancer(Anna)
1.5	¬smokes(Bob) ∨ cancer(Bob)
1.1	¬smokes(Anna) ∨ ¬friends(Anna, Anna) ∨ smokes(Anna)
1.1	¬smokes(Anna) ∨ ¬friends(Anna, Bob) ∨smokes(Bob)
1.1	¬smokes(Bob)∨ ¬friends(Bob, Anna) ∨smokes(Anna)
1.1	¬smokes(Bob) ∨ ¬friends(Bob, Bob) ∨ smokes(Bob)

Table 2.3: Full grounding of the friends & smokers MLN for the constants $Anna$ and Bob. The gray clauses and atoms can be dropped due to given evidence.

predicates have been eliminated due to evidence. In practical implementations, we do not need to store the whole set $\mathcal{G}^{\mathsf{all}}$. We can directly check if a clause can be simplified or dropped due to evidence after its generation with substitution. Section 4.1 leverages relational database management systems for efficient grounding. Example 5 illustrates the grounding on the friends & smokers example.

Example 5. *We recall the MLN of Example 3. Let us again assume that there are* 2 *constants* Anna *and* Bob*. Let us also assume that there is evidence that* Anna *is a smoker and that* Anna *and* Bob *are mutual friends:*

$$e_1 := \mathsf{smokes}(\mathsf{Anna}),$$
$$e_2 := \mathsf{friends}(\mathsf{Anna}, \mathsf{Bob}), e_3 := \mathsf{friends}(\mathsf{Bob}, \mathsf{Anna})$$

Then, we compute the set $\mathcal{G}^{\mathsf{all}}$ by substituting every variable with every constant in every clause. This results in 8 *ground atoms and in* 14 *ground clauses shown in Table 2.3.*

Due to the given evidence, we can reduce the number of clauses to 9*. Furthermore, the number of ground atoms can be reduced from* 8 *to* 5 *due to given evidence. The resulting ground Markov logic network would consist of* 5 *variables and* 9 *features. The dropped clauses and literals are highlighted in gray in Table 2.3.*

2.4 Integer Linear Programming (ILP)

In order to compute a maximum a-posteriori state of a Markov logic network, we formulate the problem as an integer linear program (ILP). We briefly recall some basic definitions and concepts of integer linear programming starting with linear programming.

Linear programming is concerned with optimizing a linear objective function over a finite number of real-valued variables, subject to a set of linear constraints over these variables. From a mathematical perspective, it can be defined as the problem of finding a point on a convex polyhedron, determined by the given linear (in-)equalities, at which the linear objective function attains a minimum or maximum [Sch99].

The canonical form of a linear program is

$$\begin{array}{lrcl} \textbf{max} & \mathbf{c}^T\mathbf{x} & & \\ \textbf{subject to} & A\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

where $\mathbf{x}$ represents a vector of variables, $\mathbf{c}$ and $\mathbf{b}$ are vectors of real-valued coefficients, and A is a matrix of coefficients. An integer linear program (ILP) is a linear program where each unknown variable is required to have integer values.

Example 6. *The integer linear program* ***max*** $0.5x + 1.0y$ ***subject to*** $x + y \leq 1$ *and* $x, y \in \{0, 1\}$ *has the solution* $x = 0$ *and* $y = 1$ *with objective value* 1.0.

Linear programs are usually solved with the simplex algorithm [Dan51]. The algorithm first constructs a feasible solution at a vertex of the polytope. Then, it traverses a sequence of vertices, such that consecutive vertices are equal or connected by a polyhedron edge and the objective function strictly improves along any traversed edge until the optimal edge is found [GK07]. Although the algorithm has a poor worst case complexity of exponential time [Min72] the average complexity is polynomial-time for most problems. It has been proven to be very efficient in practice [Sch99].

In integer linear programs one of the most common solving techniques are branch-and-bound algorithms. Thereby, we first apply the simplex algorithm on the problem. If the solution is integer, we found an optimal integer solution. If not, we choose one variable that has a non-integer value (e.g. $x = 3.3$) and restrict that variable to the next lower integer value ($x <= 3$) for one problem and to the next higher integer value for the other ($x >= 4$). This process is then repeated on each of the sub-problems. Thus, we partition the problem into smaller sub-problems. If a sub-problem undercuts the costs of a known feasible solution (maximization problem) it is excluded from all further partitionings. The challenge is to choose the optimal partition strategy. We refer the interested reader to [LW66] for a good survey of strategies. The complexity of ILP problems is NP-COMPLETE [Sch99].

Since many real world problems like for instance in the area of operations research [HLH90] can be translated to (integer) linear problems, there is the need

of very efficient solving techniques. Thus, there have been exhaustive studies on optimizing algorithms and implementing very efficient and stable solvers.

2.5 Description Logic (DL)

Formally, each description logic is a subset of first-order logic. The reason for restricting the expressivity of first-order logic is to decrease the theoretical complexity of reasoning tasks. As first-order logic, description logics (DL) formalize an application domain. This domain is described with concepts, which correspond to unary predicates, properties (also called roles), which denote binary predicates, and individuals (or instances), which can be seen as first-order logic constants. Concepts, roles, and individuals are also called entities. One concrete formalization is called ontology. From a set-theoretic interpretation, a concept is interpreted as a set of individuals and a property is interpreted as a set of pairs of individuals. Compared to databases, the domain of interpretation can be infinite and the open world assumption holds. The latter one assumes that facts which are not known to be true and not known to be false are unknown [NB03].

Tim-Berners-Lee's idea of a web of data that can be processed by machines, which is also referred to as Semantic Web [BLHL+01], has given description logics a new impetus. In order to enable researchers to build applications which are able to find implicit consequences of its explicitly represented knowledge, the World Wide Web Consortium (W3C) published the Web Ontology Language (OWL) [MVH+04] which is based on the description logic $\mathcal{SHOIN}^{(\mathcal{D})}$. Although the reasoning complexity of OWL is lower than for full first-order logic it is still intractable for larger ontologies. Thus, most applications restricted themselves to less powerful description logics. Later, the most important ones have been implemented as profiles in the new OWL 2 standard [MPSP+09, MGH+09].

In the past years researchers developed a huge variety of different description logics. For easier categorization researchers developed a naming convention which describes the operators which are allowed in the respective logic. For example, the already mentioned DL $\mathcal{SHOIN}^{(\mathcal{D})}$ follows this naming convention. In the following, we use these naming conventions without further explanation. We refer the interested reader to [NB03] for further details.

In Part II of this thesis, we combine description logics with Markov logic to log-linear description logics. The logical component of the presented theory is based on description logics for which consequence-driven reasoning is possible. We forward the reader to Section 2.5.2 for an introduction of consequence-driven reasoning and to Section 8.1 for an overview of other description logics which support consequence-driven reasoning and thus can be transferred to log-linear description logics in a straight-forward way. One of these description logics is $\mathcal{EL}++$ which we use in this thesis to explain the idea of log-linear description logics.

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\varnothing$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
GCI	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
RI	$r_1 \circ ... \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ ... \circ r_k^{\mathcal{I}} \subseteq r^{\mathcal{I}}$
RR	$\mathsf{ran}(r) \sqsubseteq C$	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C^{\mathcal{I}}$

Table 2.4: The DL $\mathcal{EL}^{++}$ with range restrictions and without concrete domains.

2.5.1 Syntax of $\mathcal{EL}^{++}$

In this section, we introduce the syntax of $\mathcal{EL}^{++}$ as one possible underlying description logic of our novel log-linear description logics. Although log-linear description logics are applicable also to more expressive DLs, we choose $\mathcal{EL}^{++}$ for two reasons. First, its expressive power is sufficient for many real world ontologies especially in the medical domain [MGH+09, BBL08]. Second, the complexity of $\mathcal{EL}^{++}$ is manageable so that it suits best for presentation purposes.

In particular, we focus on the DL $\mathcal{EL}^{++}$ *without concrete domains*, henceforth denoted as $\mathcal{EL}^{++}$. The following definitions and explanations are mainly taken from [BBL05a, BBL08, NB03, NNS11].

Concept and role descriptions in $\mathcal{EL}^{++}$ are defined recursively beginning with a set $\mathsf{N_C}$ of concept names, a set $\mathsf{N_R}$ of role names, and a set $\mathsf{N_I}$ of individual names, and are built with the constructors depicted in the column "Syntax" of Table 2.4. We write a and b to denote individual names; r and s to denote role names; and C and D to denote concept descriptions. In the following, we often omit the term *names* and will just speak of e.g. concepts, individuals, and roles.

A constraint box (CBox) is a finite set of general concept inclusion (GCI), role inclusion (RI), and range restriction (RR) axioms. Domain restriction axioms can be expressed with a GCI of the form $\exists r.\top \sqsubseteq C$ and thus are not included in Table 2.4.

A finite set of GCIs without any RIs is called a TBox. A GCI is defined as $C \sqsubseteq D$ and is constructed from two concept descriptions C and D. Each of these concept descriptions are recursively build by replacing them with $\top$, $\bot$, $\{a\}$, $C \sqcap D$, $\exists r.C$, or a plain concept name from $\mathsf{N_C}$. A RI is of the form $r_1 \circ ... \circ r_k \sqsubseteq r$ with $r, s \in \mathsf{N_R}$. We allow role inclusion axioms where $k = 0$ written as $\epsilon \sqsubseteq r$. With these axioms, we can express reflexive roles. The RR has the form $\mathsf{ran}(r) \sqsubseteq C$.

Given a CBox $\mathcal{C}$, we use $\mathsf{BC}_{\mathcal{C}}$ to denote the set of basic concept descriptions, that is, the smallest set of concept descriptions consisting of the top concept $\top$, all

concept names used in $\mathcal{C}$, and all nominals $\{a\}$ appearing in $\mathcal{C}$. A CBox $\mathcal{C}$ is in normal form if all GCIs have one of the following forms, where $C_1, C_2 \in \mathsf{BC}_\mathcal{C}$ and $D \in \mathsf{BC}_\mathcal{C} \cup \{\bot\}$:

$$C_1 \sqsubseteq D;$$
$$C_1 \sqsubseteq \exists r.C_2;$$
$$C_1 \sqcap C_2 \sqsubseteq D;$$
$$\exists r.C_1 \sqsubseteq D;$$

and if all role inclusions are of the form

$$r \sqsubseteq s;$$
$$r_1 \circ r_2 \sqsubseteq s.$$

By applying a finite set of rules and introducing new concept and role names, any CBox $\mathcal{C}$ can be turned into a normalized CBox of size polynomial in $\mathcal{C}$. We forward the reader to Section 8.3.1 for details. For any $\mathcal{EL}^{++}$ CBox $\mathcal{C}$ we write $\mathsf{norm}(\mathcal{C})$ to denote the set of normalized axioms that result from the application of the normalization rules to $\mathcal{C}$.

In Example 7 we present a small example ontology which we reuse in Part II, when we dive into log-linear description logics. The ontology has already been introduced in our informal introduction of description logics in Section 1.1.3 and is visualized in the left part of Figure 1.2.

Example 7. *Let us consider the following small example ontology $\mathcal{O}_1$.*

$Cat \sqsubseteq Animal$	*A cat is an animal.*
$Animal \sqcap Brand \sqsubseteq \bot$	*Something can not be an animal and a brand.*
$Jaguar \sqsubseteq Cat$	*A Jaguar is a cat.*
$Jaguar \sqsubseteq Brand$	*A Jaguar is a brand.*

There is a contradiction in this ontology, because a jaguar is supposed to be an animal and a brand although the concepts $Animal$ and $Brand$ are disjoint and Cat is known to be a subclass of $Animal$. The CBox of this example is $\mathcal{C} = \{Cat \sqsubseteq Animal, Animal \sqcap Brand \sqsubseteq \bot, Jaguar \sqsubseteq Cat, Jaguar \sqsubseteq Brand\}$. All GCIs are in normal form.

2.5.2 Semantics of $\mathcal{EL}^{++}$

The semantics are defined in terms of an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$. An interpretation is an assignment of elements from the real world to symbols of a description logic. $\Delta^\mathcal{I}$ is the non-empty domain of the interpretation and $\cdot^\mathcal{I}$ is the interpretation function which assigns to every $A \in \mathsf{N_C}$ a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, to every $r \in \mathsf{N_R}$ a relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and to every $a \in \mathsf{N_I}$ an element $a^\mathcal{I} \in \Delta^\mathcal{I}$.

In the following, basic naming conventions of different axiom types are discussed with intuitive examples. A concept C is subsumed by a concept D with respect to a CBox $\mathcal{C}$, written $C \sqsubseteq_\mathcal{C} D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$ in every model of $\mathcal{C}$. In the following, we often write $\sqsubseteq$ for $\sqsubseteq_\mathcal{C}$. For instance, the expression $Cat \sqsubseteq Animal$

intuitively means that every Cat is also an $Animal$. If a concept C is disjoint with a concept D (and vice versa), written $C \sqcap D \sqsubseteq \bot$, it means that no instance can be assigned to both classes C and D. In our example, the class $Animal$ is disjoint with the class $Brand$. We say that a concept name C is the domain of a role name r written as $\exists r.\top \sqsubseteq C$, if $\langle x, y\rangle \in r^{\mathcal{I}}$ implies $x \in C^{\mathcal{I}}$. For example, $\exists drives.\top \sqsubseteq Human$ states that $Human$ is the domain of $drives$. Similarly, a concept C is the range of a role name r, written as $\mathsf{ran}(r) \sqsubseteq C$, if $\langle x, y\rangle \in r^{\mathcal{I}}$ implies $y \in C^{\mathcal{I}}$. In more expressive description logics range axioms are also often represented as $\top \sqsubseteq \forall r.C$. For example, we can express that concept Car is the range of role $drives$ using the term $\mathsf{ran}(drives) \sqsubseteq Car$. Please note that although range axioms are not part of the $\mathcal{EL}^{++}$ definition, Section 8.3.1 shows how we can integrate them into $\mathcal{EL}^{++}$ and thus into log-linear $\mathcal{EL}^{++}$.

With role inclusion axioms, many important dependencies between properties can be modeled. In $\mathcal{EL}^{++}$ role hierarchies can be expressed as $r \sqsubseteq s$, transitive roles are represented with $r_1 \circ r_2 \sqsubseteq s$, and reflexive roles are constructed by setting $k = 0$ which is written as $\epsilon \sqsubseteq s$.

An interpretation $\mathcal{I}$ satisfies an axiom c if the condition in the column "Semantics" in Table 2.4 holds for that axiom. The combination of explicitly defined axioms which lead to the derivation of this axiom c is called an explanation. An interpretation $\mathcal{I}$ is a model of a CBox $\mathcal{C}$ if it satisfies every axiom in $\mathcal{C}$. A normalized CBox is classified when subsumption relationships between *all* concept names are made explicit. A CBox $\mathcal{C}$ is coherent if for all concept names C in $\mathcal{C}$ we have that $C \not\sqsubseteq_{\mathcal{C}} \bot$. If a CBox is not coherent we call it incoherent. For every axiom c and every set of axioms $\mathcal{C}'$, we write $\mathcal{C} \models c$ if every model of $\mathcal{C}$ is also a model of $\{c\}$ and we write $\mathcal{C} \models \mathcal{C}'$ if $\mathcal{C} \models c'$ for every $c' \in \mathcal{C}'$.

For a finite set $\mathsf{N_U} \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ of concept and role names the set of all normalized axioms constructible from $\mathsf{N_U}$ is the union of (a) all normalized GCIs constructible from concept and role names in $\mathsf{N_U}$ and the top and bottom concepts; and (b) all normalized RIs constructible from role names in $\mathsf{N_U}$. We refer to a pair consisting of a CBox and an ABox as ontology. An ontology $(\mathcal{C}, \mathcal{A})$ is said to be coherent if $\mathcal{C}$ is coherent.

Example 8. *Ontology $\mathcal{O}_1$ defined in Example 7 is incoherent, since the knowledge base satisfies $Jaguar \sqsubseteq \bot$. The explanation for this is that:*

$Jaguar \sqsubseteq Cat$	*(1)*
$Cat \sqsubseteq Animal$	*(2)*
$Jaguar \sqsubseteq Brand$	*(3)*
$Animal \sqcap Brand \sqsubseteq \bot$	*(4)*

We can infer from (1) and (2) that $Jaguar \sqsubseteq Animal$ (5). In axiom (4) we can replace the more specific concept name $Animal$ with $Jaguar$ due to (5) and $Brand$ with $Jaguar$ because of (3). This then leads to $Jaguar \sqcap Jaguar \sqsubseteq \bot$ which is equivalent to $Jaguar \sqsubseteq \bot$.

Reasoning services in $\mathcal{EL}^{++}$ such as consistency and instance checking can be

performed in polynomial time. Most modern reasoners use model-building methods such as optimized tableau reasoning [HST00] or extensions like hyper tableau reasoning [MSH07] for inference in description logics. For satisfiability checking tableau reasoning systematically constructs a representation model by applying a finite set of tableau rules (tableau is the french name for table). One rule always consists of a premise and a conclusion. If the premise is given, then the conclusion is derived. A set of such rules is called a tableau calculus. The main principle of tableau rules is to break complex constructs into smaller ones until complementary literals are produced or no further deduction is possible. If complementary literals are produced it is concluded that this set of formulas is unsatisfiable [MH09].

In contrast to tableau reasoning, consequence-driven reasoning derives logical consequences of axioms using inference rules. In particular, a set of rules are iteratively applied until no new logical consequences can be inferred. During this procedure all implied subsumption relations are produced. Thus, after reasoning the ontology is materialized. Since the used rules derive consequences of given axioms, such procedures are referred to as consequence-based rules [SKH11, BS13].

Part I

Efficient Inference in Markov Logic

Not everything that can be counted counts,
and not everything that counts can be counted.
(Albert Einstein)

Chapter 3

Maximum A-Posteriori Inference in Markov Logic With Integer Linear Programs

In this part of the thesis, we focus on improving the efficiency of maximum a-posteriori inference in Markov logic. The maximum a-posteriori (MAP) query computes the most probable world given evidence. Details and a formal definition of the MAP query are given in Section 2.1.3. In particular, we apply integer linear programming for solving MAP queries. Integer linear programming is a very established possibility to solve optimization problems by maximizing a given linear objective subject to linear constraints. Section 2.4 formally introduces integer linear programming.

The advantages of utilizing ILP compared to traditional approaches for solving weighted SAT problems are that ILP is a very established and exhaustively studied research field and there exist numerous efficient ILP solvers. Moreover, ILP solvers guarantee a certain solution quality.

Consequently, integer linear programming has been used in many tasks to solve MAP problems [KPRY05, CL07]. For Markov logic, Riedel [Rie08] has proposed a translation of MLNs to ILPs. His translation was inspired form previous work in the area of mathematical programming [Wil99]. Compared to the general translation of Markov networks to ILP [TCKG05] the translation proposed by Riedel is more compact. Another translation was proposed in the context of max-margin weight learning for MLNs [HM09] where the MAP query was formulated as a linear relaxation of an ILP and a rounding procedure was applied to extract an approximate MAP state. Section 3.1 discusses the translation of Riedel [Rie08] which needs three constraints for each soft formula.

Section 3.2 and Section 3.4 provide the answer to research question Q1 of Section 1.2.1:

Q1 *Can we improve existing ILP translation techniques such that we reduce the size of the ILP and make the symmetries of the model more explicit to sym-*

metry detection heuristics?

In Section 3.2, we will provide a new and even more compact ILP translation of MAP queries for Markov logic. This translation is novel, since it requires only one linear constraint per ground clause, irrespective of the ground clause being weighted or unweighted.

This ILP translation can still be optimized such that grounding the whole Markov network becomes unnecessary. In this context, the concept of cutting plane inference (CPI) [Rie08], which solves multiple smaller ILPs in several iterations, has shown remarkable performance increases. In the CPI algorithm, we start with the given evidence and add only the violated constraints. Then, we compute an intermediate solution and again add the violated constraints with respect to this solution. This process is repeated until no violated constraint remain. Riedel's cutting plane inference algorithm is explained in Section 3.3.

In Section 3.4 we present the major contribution of Part I which we call cutting plane aggregation (CPA). The CPA method aggregates ground clauses by exploiting symmetries. Contrary to existing symmetry exploitation and lifted inference algorithms (see Section 5.1) that can cope with no or only a limited amount of evidence, the presented approach specifically exploits symmetries induced by evidence. Moreover, the model symmetry is exploited by presenting it to ILP solvers in such a way that the solvers' internal symmetry detection algorithms can take advantage of it.

Furthermore, we provide an algorithm that combines our cutting plane aggregation approach with Riedel's cutting plane inference method in Section 3.5.1. Combining the CPA with the CPI algorithm results in a lower number of ILP constraints than CPI or CPA separately. Additionally, the parallelization presented in Section 3.5.2 exploit multi-core architectures. Both leads to lower runtimes as we will show in our experimental section. The parallelization algorithm of Section 3.5.2 answers the first part of the research question Q2:

Q2 *How can we parallelize the solution process and tightly integrate relational database management systems (RDBMS) within this process?*

Section 3.2, Section 3.4, and Section 3.5 is our work [NNS13]. Some text passages, definitions, tables, and graphics are taken from our publication [NNS13]. We extended our cutting plane aggregation algorithm to work also with hard constraints. Section 3.1 and Section 3.3 present a revised version of [Rie08].

3.1 Standard Integer Linear Program Translation

We start with the standard ILP translation of the MAP query as proposed by [Rie08, Wil99]. As input we require a set of ground clauses $\mathcal{G}$ whose construction is explained in Section 2.3.2.

For transforming the MAP query into an ILP, we associate one binary ILP variable $x_\ell \in \{0, 1\}$ with each ground atom ℓ occurring in the set of ground clauses.

For a ground clause g let $L^+(g)$ be the set of ground atoms occurring unnegated in g and $L^-(g)$ be the set of ground atoms occurring negated in g.

First, we encode the given evidence by introducing linear constraints of the form $x_\ell \leq 0$ or $x_\ell \geq 1$ depending on whether the evidence set the corresponding ground atom ℓ to false or true.

For every ground clause $g \in \mathcal{G}$ with weight $w_g \neq \infty$, we add a novel binary variable z_g and add the following constraints to the ILP:

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \geq z_g.$$

$$x_\ell \leq z_g \quad \forall\, \ell \in L^+(g)$$
$$(1 - x_\ell) \leq z_g \quad \forall\, \ell \in L^-(g)$$

The first constraint ensures that the binary variable z_g is zero if all positive (negative) literals ℓ of g are *false* (*true*). The constraints in the second line force the binary variable z_g to be one if one non-negated literal ℓ is true while the latter constraints force the binary variable z_g to be one if one negated literal ℓ is false, respectively.

The number of ILP constraints needed for one clause g amounts $1 + |L^+(g)| + |L^-(g)|$. If one clause contains for instance 3 literals, the number of constraints needed is 4.

For every g with weight $w_g = \infty$, that is, a hard clause, we add the following linear constraint to the ILP

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \geq 1.$$

Finally, the objective of the ILP is

$$\max \sum_{g \in \mathcal{G}} w_g z_g,$$

where w_g is the weight of g and $z_g \in \{0, 1\}$ is the binary variable previously associated with ground clause g. We compute a MAP state by solving the ILP whose solution corresponds one-to-one to a MAP state $\mathbf{x}$ where $x = true$ if the corresponding ILP variable is 1 and $x = false$ otherwise.

Table 3.1 depicts three example clauses and the respective ILP formulations. It visualizes that we need more than one ILP constraint for each soft clause in this standard translation. In the next section we present an optimized translation where we need at most one ILP constraint for every clause.

3.2 Optimized Integer Linear Program Translation

We now improve the standard ILP translation presented in Section 3.1 so that it requires only one linear constraint per ground formula. In order to transform the

weight	ground clause		**max** $1.1z_1 - 0.5z_2$ **subject to**
1.1	$x_1 \vee \neg x_2 \vee x_3$		$x_1 + (1 - x_2) + x_3 \geq z_1$
			$x_1 \leq z_1$
			$(1 - x_2) \leq z_1$
		$\rightsquigarrow$	$x_3 \leq z_1$
-0.5	$\neg x_1 \vee x_2$		$(1 - x_1) + x_2 \geq z_2$
			$(1 - x_1) \leq z_2$
			$x_2 \leq z_2$
∞	$\neg x_1 \vee x_2$		$(1 - x_1) + x_2 \geq 1$

Table 3.1: An example of the standard ILP transformation (inspired by [NNS13]). We need multiple constraints for each soft clause.

MAP query problem to an ILP we again have to first ground, that is, instantiate, the first-order theory specified by the Markov logic network. We refer to Section 2.3.2 for details. After grounding, we receive the set $\mathcal{G}$.

We again associate one binary ILP variable x_ℓ with each ground atom ℓ occurring in the set of ground clauses. For a ground clause g let $L^+(g)$ be the set of ground atoms occurring unnegated in g and $L^-(g)$ be the set of ground atoms occurring negated in g.

As before, we encode the given evidence by introducing linear constraints of the form $x_\ell \leq 0$ or $x_\ell \geq 1$ depending on whether the evidence set the corresponding ground atom ℓ to false or true.

For every ground clause $g \in \mathcal{G}$ with weight $w_g > 0,\ w_g \in \mathbb{R}$, we add a novel binary variable z_g and the following constraint to the ILP:

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \geq z_g.$$

Because the ILP aims to maximize the objective and because the weight w is greater than zero, the variable z_g will always try to become as large as possible. Thus, we do not need any constraints for forcing the variable to become one. We just need the above constraint which ensures that z_g has to be zero if the ground clause g is false in the current truth assignment.

For every g with weight $w_g < 0,\ w \in \mathbb{R}$, we add a novel binary variable z_g and the following constraint to the ILP:

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \leq (|L^+(g)| + |L^-(g)|) z_g.$$

In case of negative weights ($w < 0$) the variable z_g *tries* to be as small as possible. Consequently, we do not need further constraints to restrict the variable to zero. The above constraint forces the variable z_g to become one if the ground clause g is true in the current truth assignment and thus is the only constraint we need. This constraint is novel and differs from the standard formulation of [Rie08, Wil99, HM09] described in Section 3.1.

weight	ground clause		**max** $1.1z_1 - 0.5z_2$ **subject to**
1.1	$x_1 \vee \neg x_2 \vee x_3$	$\rightsquigarrow$	$x_1 + (1 - x_2) + x_3 \geq z_1$
-0.5	$\neg x_1 \vee x_2$		$(1 - x_1) + x_2 \leq 2 \cdot z_2$
∞	$\neg x_1 \vee x_2$		$(1 - x_1) + x_2 \geq 1$

Table 3.2: An example of the optimized ILP transformation [NNS13]. We just need one constraint for each soft clause.

If a ground clause has zero weight we do not have to add the corresponding constraints.

In case of hard clauses with weight $w_g = \infty$ the transformation stays as before. We add the following linear constraint to the ILP:

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \geq 1$$

Finally, the objective of the ILP is:

$$\max \sum_{g \in \mathcal{G}} w_g z_g,$$

where w_g is the weight of g and $z_g \in \{0, 1\}$ is the binary variable previously associated with ground clause g.

Table 3.2 depicts three clauses and the respective ILP formulations for $w > 0$, $w < 0$, and $w = \infty$. Again, note that the presented transformation is more efficient than the one described in [Rie08, Wil99] since it only requires one constraint per ground clause instead of three. To that end, we encourage the reader to compare this table with Table 3.1.

Currently, we assumed that we have to include every ground clause in the set $\mathcal{G}$ in our ILP. In real world problems, this often leads to a very large amount of clauses resulting in oversized and often intractable integer linear programs. In the next Section, we will focus on a concept called cutting plane inference, which often leads to a significant smaller number of required constraints.

3.3 Cutting Plane Inference (CPI)

Cutting plane methods for ILPs begin by solving the linear relaxation of the given ILP, that is, the ILP without the requirement that the solution has to be integer. The approach proceeds to test whether the computed solution is integer. If it is not, there exists a linear inequality separating the optimal objective value from the convex hull of the true feasible set. This inequality is called a cut. A cut can be added as a constraint to the linear program which is then resolved. This process is repeated until a complete integer solution is found.

While the basic idea is similar to that from operations research for solving integer linear programs, the cutting plane inference (CPI) approach for Markov

logic [Rie08] is a meta-algorithm operating between the grounding algorithm and the ILP solver. Instead of immediately adding one constraint for each ground clause to the ILP formulation, the ILP is initially formulated so as to enforce the evidence to hold in any solution. Based on the solution of this more compact ILP one determines the violated constraints, adds these to the ILP, and solve the ILP again. This process is repeated until no constraints are violated by an intermediate solution.

This can often greatly reduce the number of linear constraints of the resulting ILP. Indeed, in numerous practical applications the application of the cutting plane inference algorithm has made previously intractable MAP queries solvable within seconds [Rie08, NMS10, NNS11].

Our CPI implementation is illustrated in Algorithm 1. It starts with the intermediate solution (Herbrand interpretation) $H^{(0)} \subseteq \mathcal{H}$ containing the evidence atoms and all axioms which belong to a ground clause consisting of one axiom only and having a positive weight. The set of ground clauses whose corresponding linear constraint have been added to the ILP are denoted by $\mathcal{G}_{\mathsf{ILP}}$. This set consist initially of the ground clauses encoding $H^{(0)}$. The set $\mathcal{G}_{\mathsf{new}}$ contains the violated constraints which are added in each CPI iteration. Line 1-4 perform this initialization.

The CPI algorithm performs the following sequence of steps in iteration t. Line 6 sets $\mathcal{G}_{\mathsf{new}}$ to the empty set. Depending on the weight of the ground clauses, we determine the violated constraints as follows (Lines 7-11):

- For every ground clause $g \in \mathcal{G} \setminus \mathcal{G}_{\mathsf{ILP}}$ with $w_g > 0$ or $w_g = \infty$ which is not satisfied by the current intermediate solution $H^{(t)}$, we add g to $\mathcal{G}_{\mathsf{new}}$; and
- For every ground clause $g \in \mathcal{G} \setminus \mathcal{G}_{\mathsf{ILP}}$ with $w_g < 0$ which is satisfied by the current intermediate solution $H^{(t)}$, we add g to $\mathcal{G}_{\mathsf{new}}$.

If no violated ground clauses were found in step 2 (which means that set $\mathcal{G}_{\mathsf{new}}$ is empty), then the current solution $H^{(t)}$ is optimal with respect to the set of all ground clauses and we are done. In this case, we jump from Line 12 to Line 18 and return the MAP state in Line 19.

Otherwise, we add exactly the linear constraints based on the set $\mathcal{G}_{\mathsf{new}}$ to the ILP in Line 14. Then, Line 15 adds every ground clause $g \in \mathcal{G}_{\mathsf{new}}$ to the set $\mathcal{G}_{\mathsf{ILP}}$. Finally, we solve this ILP obtaining the next intermediate solution $H^{(t+1)}$ in Line 16, and proceed again with Line 6.

Constraints corresponding to violated ground clauses are added to the ILP. In many cases, this iterative addition of constraints leads to ILPs with far less constraints than the ILP that would result from naively adding all constraints.

The algorithm for determining the ground clauses violated in an intermediate solution in Step 1 must be very efficient as the benefit of more compact ILPs would be neutralized otherwise. Based on existing work [Rie08], we use a relational database management system to compute the violated clauses. The computation of the violated clauses can be formulated as a sequence of join queries. Details are described in Section 4.2.

Algorithm 1 Cutting plane inference (CPI) algorithm.

Input: $\mathcal{G}$: Set of all ground clauses
Input: **E**: Set of all evidence axioms
Output: H^*: Maximum a-posteriori state
PERFORMCPI

$\mathcal{G}_{\mathsf{ILP}} \leftarrow$ **E**$\cup$ all $g \in \mathcal{G}$ having one literal only and a weight $w_g > 0$.
Initial solution $H^{(0)} \leftarrow$ all atoms in $\mathcal{G}_{\mathsf{ILP}}$.
transfer each $g \in \mathcal{G}_{\mathsf{ILP}}$ to linear constraints and add them to the ILP.
$t \leftarrow 0$
repeat
  $\mathcal{G}_{\mathsf{new}} \leftarrow \varnothing$
  for every ground clause $g \in \mathcal{G} \setminus \mathcal{G}_{\mathsf{ILP}}$ **do**
    if (weight $w_g > 0$ or $w_g = \infty$ **and** g is **not** satisfied with $H^{(t)}$) **or** (weight $w_g < 0$ **and** g is satisfied with $H^{(t)}$) **then**
      add g to $\mathcal{G}_{\mathsf{new}}$.
    end if
  end for
  if $\mathcal{G}_{\mathsf{new}} \neq \varnothing$ **then**
    $t \leftarrow t + 1$
    transfer each $g \in \mathcal{G}_{\mathsf{new}}$ to linear constraints and add them to the ILP.
    add every $g \in \mathcal{G}_{\mathsf{new}}$ to the set $\mathcal{G}_{\mathsf{ILP}}$.
    solve the ILP and set $H^{(t)}$ to the solution of the ILP.
  end if
until $\mathcal{G}_{\mathsf{new}} = \varnothing$
return $H^{(t)}$

Example 9. *Let us revisit the friends & smokers MLN from Example 3. Let us again assume that there are* 2 *constants* Anna *and* Bob*. Let us also assume that there is evidence that* Anna *is a smoker and that* Anna *and* Bob *are mutual friends:*

$$g_1 := \mathsf{smokes}(\mathsf{Anna}),$$
$$g_2 := \mathsf{friends}(\mathsf{Anna}, \mathsf{Bob}), g_3 := \mathsf{friends}(\mathsf{Bob}, \mathsf{Anna})$$

In Example 5 we already noted that the naive grounding of the MLN would lead to 14 *clauses and* 8 *ground atoms. If we consider evidence, we result in* 9 *ground clauses and* 5 *ground atoms. Now, let us apply the cutting plane inference algorithm. We have that* $H^{(0)} = \{g_1, g_2, g_3\}$ *and* $\mathcal{G}_{\mathsf{ILP}} = \{g_1, g_2, g_3\}$*. We proceed to determine the violated ground clauses given* $H^{(0)}$*:*

(1) $\neg\mathsf{smokes}(\mathsf{Anna})$
(2) $\neg\mathsf{friends}(\mathsf{Anna}, \mathsf{Bob})$
(3) $\neg\mathsf{friends}(\mathsf{Bob}, \mathsf{Anna})$
(4) $\neg\mathsf{smokes}(\mathsf{Anna}) \vee \mathsf{cancer}(\mathsf{Anna})$
(5) $\neg\mathsf{smokes}(\mathsf{Anna}) \vee \neg\mathsf{friends}(\mathsf{Anna}, \mathsf{Bob}) \vee \mathsf{smokes}(\mathsf{Bob})$

Only the ground clauses (4) and (5) have to be added to $\mathcal{G}_{\mathsf{ILP}}$ *since all others consist solely of evidence atoms. The ILP is updated and resolved, obtaining* $H^{(1)} = \{g_1, g_2, g_3, g_4, g_5\}$ *with* $g_4 := \mathsf{smokes}(\mathsf{Bob})$ *and* $g_5 := \mathsf{cancer}(\mathsf{Anna})$*. Again, the violated ground clauses in* $\mathcal{G} \setminus \mathcal{G}_{\mathsf{ILP}}$ *are determined:*

(1') $\neg\mathsf{smokes}(\mathsf{Bob})$
(2') $\neg\mathsf{smokes}(\mathsf{Bob}) \vee \mathsf{cancer}(\mathsf{Bob})$
(3') $\neg\mathsf{cancer}(\mathsf{Anna})$

All of these ground clauses are added to $\mathcal{G}_{\mathsf{ILP}}$*, the ILP is updated accordingly and resolved, obtaining* $H^{(2)} = \{g_1, g_2, g_3\}$*. There are no violated ground clauses and, therefore, the current solution is optimal and* $H^{(2)}$ *is a MAP state. Thus, we only had to translate 5 clauses to the ILP plus 3 clauses which model the given evidence.*

The previous example demonstrates that the number of clauses which need to be translated to ILP constraints is often much smaller when we use the CPI method. The naive grounding without evidence consideration leads to 14 clauses, grounding with evidence consideration results in 9 clauses, and applying the CPI algorithm lead to 5 clauses, a bit more than one third of the clauses of the naive formulation. In all three cases, we have to add 3 more constraints to model the given evidence. Please note, that in larger problems with more constraints and more evidence axioms, this reduction is usually larger. We forward to our experiments in Chapter 6 for details.

We now turn to the novel cutting plane aggregation method, the main contribution of Part I of this thesis. The cutting plane aggregation approach bundles clauses of the ground MLN that exhibit symmetry during the CPI phase. This allows us to reduce not only the number of constraints, but also to enable internal symmetry detection mechanisms of ILP solvers to work more efficiently by detecting symmetry in the resulting constraints.

3.4 Cutting Plane Aggregation (CPA)

In this section, we further optimize the translation of ground clauses to ILP constraints. More concretely, we introduce a novel approach that aggregates sets of constraints so as to make the resulting ILP have (a) fewer variables (b) fewer constraints and (c) its symmetries more exposed to the symmetry detection approaches implemented in ILP solvers. Compared to [NNS13], we additionally aggregate hard constraints.

We first demonstrate that evidence often introduces symmetries in the resulting sets of ground clauses and, therefore, at the feature level. The proposed approach aggregates ground clauses exposing this type of symmetry to symmetry detection algorithms of the ILP solvers. These algorithms apply heuristics to test whether the constraint matrix of the ILP exhibits symmetries in form of permutations on its variables. For an overview of existing principles and algorithms for detecting

g	ℓ_i	c	w
g_1	$x_1 \vee$	$\neg y_1 \vee y_2$	1.0
g_2	$x_2 \vee$	$\neg y_1 \vee y_2$	1.0
g_3	$\neg x_3 \vee$	$\neg y_1 \vee y_2$	1.0
g_4	$\neg x_4 \vee$	$\neg y_1 \vee y_3$	1.0
g_5	$\neg x_5 \vee$	$y_1 \vee y_2$	1.0
g_6	$x_6 \vee$	$\neg y_1$	∞

$\rightsquigarrow$

ℓ_i	c	w
$x_1 \vee$		
$x_2 \vee$	$\neg y_1 \vee y_2$	1.0
$\neg x_3 \vee$		
$\neg x_4 \vee$	$\neg y_1 \vee y_3$	1.0
$\neg x_5 \vee$	$y_1 \vee y_2$	1.0
$x_6 \vee$	$\neg y_1$	∞

Table 3.3: An example of ground clauses that can be aggregated [NNS13].

and exploiting symmetries in integer linear programs, we forward the reader to Section 3.4.1.

We describe the cutting plane aggregation (CPA) approach in two steps. First, we explain how the ground formulas are aggregated and, second, we discuss how these aggregated ground formula are transformed to ILP constraints.

Definition 1. *Let $G \subseteq \mathcal{G}$ be a set of n weighted ground clauses and let c be a ground clause. Let weights be in $\mathbb{R} \cup \infty$. We say that G can be aggregated with respect to c if (a) all ground clauses in G all have the same weight and (b) for every $g_i \in G, 1 \leq i \leq |G|$, we have that $g_i = \ell_i \vee c$ where ℓ_i is a (positive or negative) literal for each $i, 1 \leq i \leq |G|$.*

Please note that this definition aggregates weighted ground clauses *and* hard ground clauses, which is an extension compared to [NN10]. Intuitively, it is not a difference for condition (a) if weight w is in the real numbers $w \in \mathbb{R}$ or infinitive $w = \infty$.

Example 10. *Table 3.3 lists a set of 6 ground clauses. The set of clauses $\{g_1, g_2, g_3\}$ can be aggregated with respect to $\neg y_1 \vee y_2$ since we can write each of these ground clauses as $\ell_i \vee \neg y_1 \vee y_2$ with $\ell_1 := x_1, \ell_2 := x_2$, and $\ell_3 := \neg x_3$.*

Before we describe the algorithmic advantages of finding ground clauses that can be aggregated and the corresponding ILP formulation of such clauses, we provide a typical instance of a Markov logic network resulting in a large number of clauses that can be aggregated. In Example 11 CPA aggregates 100 clauses to only 1 clause by considering evidence.

Example 11. *Let us revisit the example MLN in Table 3 and consider the clause $\neg\mathsf{smokes}(x) \vee \mathsf{cancer}(x)$. Let us assume that there are 100 constants $C_1, ..., C_{100}$ for which we have evidence $\mathsf{smokes}(C_i), 1 \leq i \leq 100$. For $1 \leq i \leq 100$, let x_i be the ILP variable corresponding to the ground atom $\mathsf{smokes}(C_i)$ and y_i be the ILP variable corresponding to the ground atom $\mathsf{cancer}(C_i)$. The initial ILP would contain the constraints $x_i \geq 1$ encoding the evidence. Due to evidence $\neg\mathsf{smokes}(C_i) \vee \mathsf{cancer}(C_i), 1 \leq i \leq 100$, would simplify to $\mathsf{cancer}(C_i), 1 \leq i \leq 100$,. Hence, the naive translation would add 100 constraints $y_i \geq z_i$ to the ILP and the objective of the ILP with respect to these clauses would be $\max 1.5z_1 + ... + 1.5z_{100}$. However,*

here we can aggregate the ground clauses $\mathsf{cancer}(C_i), 1 \leq i \leq 100$, *for* c = *false and* $\ell_i = \mathsf{cancer}(C_i), 1 \leq i \leq 100$.

Detecting the optimal clauses c with which a set of ground clauses $\mathcal{G}$ can be aggregated is not a trivial problem. An optimal strategy which minimizes the number of clauses c is computational expensive. Thus, we suggest an approximate strategy. The strategy is presented in Section 3.4.2.

The following lemma provides the basis of a novel ILP translation for aggregated ground clauses, that is more efficient and allows the ILP solver to detect the inherent symmetry introduced by the evidence.

Lemma 2. *Let* $G \subseteq \mathcal{G}$ *be a set of ground clauses with weight* w *and let* c *be a ground clause. Moreover, let us assume that* G *can be aggregated with respect to* c*, that is, that each* $g \in G$ *can be written as* $\ell_i \vee c$*. The aggregated feature* f^G *for the aggregated clauses* G *with weight* w *maps each interpretation* I *to an integer value as follows*

$$f^G(I) = \left\{ \begin{array}{cc} |G| & if \;\; I \models c \\ |\{\ell_i \vee c \in G \mid I \models \ell_i\}| & otherwise \end{array} \right\}.$$

For each set of ground clauses $G \subseteq \mathcal{G}$ *with weight* $w = \infty$*, we require* $f^G(I) > |G|$

Proof. An individual feature $f^g(I)$ is defined as

$$f^g(I) = \left\{ \begin{array}{ll} 1 & if \; I \models g \\ 0 & otherwise \end{array} \right. .$$

Since all $g \in G$ share the same weight, we can aggregate the features to an aggregated feature by

$$f^G(I) = \sum_{g \in G} f^g(I).$$

Now we are in the position to distinguish two cases:

$I \models c$: Since each $g \in G$ can be written as $\ell_i \vee c$, each g is satisfied if $I \models c$. Thus, each individual feature function returns $f^g(I) = 1$. Consequently, the aggregated feature function has to return $f^G(I) = \sum_{g \in G} f^g(I) = |G|$.

$I \not\models c$: If $I \not\models c$ each clause $g \in G$ equals $g \equiv \ell_i \vee false \equiv \ell_i$. Thus, the individual feature function $f^g(I)$ of each clause g returns

$$f^g(I) = \left\{ \begin{array}{ll} 1 & if \; I \models \ell_i \\ 0 & otherwise \end{array} \right. .$$

Finally, we rewrite the aggregated feature function as

$$f^G(I) = \sum_{g \in G} f^g(I) = |\{\ell_i \vee c \in G \mid I \models \ell_i\}|.$$

ℓ_i	c	w		**max** $0.5z_1 - 1.5z_2$ **subject to**
$x_1 \vee$ $x_2 \vee$ $x_3 \vee$	y_1	0.5		$x_1 + x_2 + x_3 + 3y_1 \geq z_1$ $z_1 \leq 3$
$\neg x_1 \vee$ $x_2 \vee$ $\neg x_3 \vee$	$y_1 \vee \neg y_2$	-1.5	$\rightsquigarrow$	$(1 - x_1) + x_2 + (1 - x_3) \leq z_2$ $3 \cdot y_1 \leq z_2$ $3 \cdot (1 - y_2) \leq z_2$
$x_1 \vee$ $\neg x_2 \vee$ $\neg x_3 \vee$	$\neg y_1$	∞		$x_1 + (1 - x_2) + (1 - x_3) + 3(1 - y_1) \geq 3$

Table 3.4: An example of the aggregated ILP formulation [NNS13]. For the sake of simplicity, we denote the ground atoms and ILP variables with identical names.

Please note, that this lemma is also valid for sets of ground clauses G containing only one g. In this case, the individual feature $f^g(I)$ equals the aggregated feature $f^G(I)$ and the choice of c is arbitrary. □

The feature resulting from the aggregation, therefore, counts the number of literals ℓ_i that are satisfied whenever the ground clause c is not satisfied and returns the number of aggregated clauses otherwise.

Please also note that an encoding of this feature as a factor in a factor graph would need space exponential in the number of ground atoms even though the feature only has a linear number of different possible values. The feature, therefore, is highly symmetric – each assignment to the random variables corresponding to the positive (negative) literals that has the same Hamming weight results in the same feature weight contribution. This constitutes a feature-specific local form of finite exchangeability [Fin72, Dia77] of random variables induced by the evidence. Therefore, we denote this form of finite exchangeability as context-specific exchangeability.

While standard models such as factor graphs cannot represent such symmetric features compactly, one can encode these counting features directly with a number of constraints that is linear in the number of aggregated clauses. We now describe this translation in more detail.

As before, for any ground clause c, let $L^+(c)$ ($L^-(c)$) be the set of ground atoms occurring unnegated (negated) in c. Let $G \subseteq \mathcal{G}$ be a set of n ground clauses with *positive* weight $w > 0, w \in \mathbb{R}$ that can be aggregated with respect to c, that is, for each $g \in G$ we have that $g = x_i \vee c$ or $g = \neg x_i \vee c$ for a ground atom x_i and a fixed clause c. We now add the following two linear constraints to the ILP:

$$\sum_{(x_i \vee c) \in G} x_i + \sum_{(\neg x_i \vee c) \in G} (1 - x_i) + \sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) \geq z_G \tag{3.1}$$

and

$$z_G \leq n \tag{3.2}$$

Linear Constraint 3.1 introduces the novel *integer* variable z_G for each aggregation. Whenever a solution satisfies the ground clause c this variable has the value n and otherwise it is equal to the number of literals ℓ_i satisfied by the solution. Since Constraint 3.1 alone might lead to values of z_G that are greater than n, the linear Constraint 3.2 ensures that the value of z_G is at most n. However, linear Constraint 3.2 only needs to be added if clause c is not the constant *false*.

We now turn to the case when we can aggregate clauses with *negative* weight. Let $G \subseteq \mathcal{G}$ be a set of n ground clauses with weight $w < 0$ that can be aggregated with respect to c, that is, for each $g \in G$ we have that $g = x_i \vee c$ or $g = \neg x_i \vee c$ for a ground atom x_i and a fixed clause c. We now add the following linear constraints to the ILP:

$$\sum_{(x_i \vee c) \in G} x_i + \sum_{(\neg x_i \vee c) \in G} (1 - x_i) \leq z_G \tag{3.3}$$

and

$$n x_\ell \leq z_G \text{ for every } \ell \in L^+(c) \tag{3.4}$$

and

$$n(1 - x_\ell) \leq z_G \text{ for every } \ell \in L^-(c) \tag{3.5}$$

Linear Constraint 3.3 introduces an integer variable z_G. Intuitively, this variable counts the number of clauses in G that are satisfied. Since the weight is negative, z_G tries to be as small as possible. With Formula 3.3 we count every satisfied ground atoms x_i in $g = x_i \vee c$ and every satisfied literal $\neg x_i$ in $g = \neg x_i \vee c$ for each $g \in G$ so that variable z_G must not become smaller than the sum of both. Equation 3.4 and Equation 3.5 ensures that $z_G \geq n$ if one of the positive or negative literals ℓ in c are satisfied. Although we result in few more constraints for negative constraints, their number is still much smaller since we usually can aggregate many ground clauses g. Furthermore, the number of constraints is constant with respect to all existing aggregated clauses G. In the formulation of Section 3.2 each of those ground clauses would require one constraint.

In case of hard clauses with *infinitive weight*, each $G \subseteq \mathcal{G}$ has weight $w = \infty$. We have to add only one linear constraint for each G which is as followed:

$$\sum_{(x_i \vee c) \in G} x_i + \sum_{(\neg x_i \vee c) \in G} (1 - x_i) + \sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) \geq n \tag{3.6}$$

In the case of aggregating hard clauses, no new integer variable is required. We only need one constraint, which ensures that each clause $g \in G$ is satisfied. If the solution satisfies one of the aggregated literals $\ell \in L^+(c)$ (and $\neg\ell \in L^-(c)$) all clauses $g \in G$ are also satisfied. In this case, the last part of the ILP constraint $\sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) \geq n$ is fulfilled. In case none of the aggregated literals $\ell \in L^+(c)$ (or $\neg\ell \in L^-(c)$) are satisfied, all literals $x_i \vee c$ (and $\neg x_i \vee c$) must be satisfied. This is covered by the first part of the restriction, which then ensures $\sum_{(x_i \vee c) \in G} x_i + \sum_{(\neg x_i \vee c) \in G} (1 - x_i) \geq n$.

For each of the integer variables z_G, which has been introduced for an aggregated set of clauses with $w_g \in \mathbb{R}$, we add the term $w_g z_G$ to the objective function

where w_g is the weight of any of the aggregated clauses. For hard clauses with infinitive weights, nothing has to be added to the objective function. If a ground clause has zero weight we do not have to add the corresponding constraints. Table 3.4 shows a set of aggregated ground clauses and the corresponding integer linear program.

Please note that the length of constraints increases when we apply our cutting plane aggregation algorithm. However, due to the lower number of variables required and due to the increased ability of ILP solvers to detect symmetries, runtimes are reduced. The interested reader is referred to Section 6.4.2 for an empirical confirmation.

We now have the following theorem.

Theorem 3. *Let $\mathcal{M}$ be a Markov logic network and ILP($\mathcal{M}$) be the ILP formulation with aggregated cutting planes. Each solution of ILP($\mathcal{M}$) corresponds one-to-one to a maximum a-posteriori state of the Markov logic network $\mathcal{M}$.*

Proof. Weighted MAX-SAT problems have been transferred to ILP formulations [BF98]. Building on these results, we have to show that our ILP translation above correctly translates the aggregated features from Lemma 2. Let $G \subseteq \mathcal{G}$ be a set of aggregated ground clauses with weight w. Moreover, let us assume that G can be aggregated with respect to c, that is, that each $g \in G$ can be written as $\ell_i \vee c$. Let I be any interpretation. For each aggregated feature $f^G(I)$ we introduced an integer ILP variable z_G and added $w \cdot z_G$ to the objective. For showing the correct translation, we have to show that $z_G = f^G(I)$.

In case of $w > 0$, the ILP constraints in Formula 3.1 and Formula 3.2 ensure that $z_G = f^G(I)$.

- If $I \models c$, then Formula 3.2 ensures that $z_G \leq |G|$. Since the ILP tries to maximize z_G and Formula 3.1 does not further restrict z_G we can conclude that $z_G = |G| = f^G(I)$.

- If $I \not\models c$, Formula 3.1 restricts z_G to $|\{\ell_i \vee c \in G \mid I \models \ell_i\}| \leq z_G$. Again, due to maximization and since Formula 3.2 does not further restrict z_G, we can conclude $|\{\ell_i \vee c \in G \mid I \models \ell_i\}| = z_G$.

In case of $w < 0$, the ILP constraints in Formula 3.3, Formula 3.4, and Formula 3.5 ensure that $z_G = f^G(I)$.

- If $I \models c$, then Formulas 3.4 and Formulas 3.5 ensure that $z_G \geq |G|$. Formulas 3.4 go through all literals $\ell \in L^+(c)$ in clause c and set $z_G \geq |G|$ if one of them is entailed in I. Similarly, Formulas 3.5 go through all negated literals $\ell \in L^-(c)$ in clause c and set $z_G \geq |G|$ if one of them is not entailed in I. Since $I \models c$, z_G is set to $z_G \geq |G|$ because at least one $\ell \in L^+(c) : I \models \ell$ or one $\ell \in L^-(c) : I \not\models \ell$. Since $w < 0$ the ILP tries to minimize z_G and Formula 3.3 does not further restrict z_G, we conclude $z_G = |G| = f^G(I)$.

- If $I \not\models c$, we obtain $z_G \geq 0$ according to Formulas 3.4 or Formulas 3.4. Then, Formula 3.3 ensures $|\{\ell_i \vee c \in G \mid I \models \ell_i\}| \geq z_G$. Again, since $w < 0$ the ILP minimizes z_G and thus $|\{\ell_i \vee c \in G \mid I \models \ell_i\}| = z_G$.

□

Example 12 shows a simple but typical example how the cutting plane aggregation approach can decrease the number of constraints and the number of required variables significantly. In this small example we can reduce the number of constraints from 100 to 1.

Example 12. *Let us revisit Example 11. We established that we can aggregate the ground clauses* $\mathsf{cancer}(C_i), 1 \leq i \leq 100$, *for* $c = false$ *and* $\ell_i = \mathsf{cancer}(C_i), 1 \leq i \leq 100$. *Now, instead of using* 100 *linear constraints and* 100 *novel summands in the objective function we add the following single linear constraint:*

$$y_1 + ... + y_{100} \leq z_G$$

and the term $1.5z_G$ *to the objective function. Not only is the representation more compact but it also allows ILP solvers to more straight-forwardly detect symmetry in the model since permutations of the variables lead to an identical ILP.*

3.4.1 Symmetry Detection in ILP

Despite the reduction of the number of ILP variables and ILP constraints, our aggregation also makes symmetries explicit for state-of-the-art ILP solvers. An ILP is defined to be symmetric, if its variables can be permuted without changing the structure of the problem [Mar03]. We start with a brief summary of the survey [Mar10] about symmetry detection in integer linear programs extended with the concept of orbit branching [OLRS11, OLRS07].

Most of the symmetry detection algorithms assume that next to the objective function and the ILP constraints a so called symmetry group is given as input. A symmetry group contains permutations of variables that lead to a similar ILP. Computing this symmetry group is an NP-COMPLETE problem. Practical algorithms focus instead on computing symmetry groups for the LP-relaxation of the original ILP problem. In the LP-relaxation, all binary constraints of variables (e.g. $x \in \{0, 1\}$) are replaced by its relaxation (e.g. $0 \leq x \leq 1$). Thus, the solution usually does not fulfill the integer requirement of ILPs. Those problems then can be mapped to colored graphs and efficiently solved with efficient graph isomorphism algorithms [ARMS03] (like for instance with the tool SAUCY [DLSM04]).

If such a symmetry group is given, there exist a vast amount of approaches, which exploit symmetries for solving ILPs. Most of them share the idea of introducing new constraints due to symmetry information that reduce the search space of possible solutions. Those constraints are called symmetry breaking inequalities since they break the symmetry of variables. We distinguish between so-called dynamic symmetry breaking inequalities and static symmetry breaking inequalities.

Static symmetry inequalities are added to the initial formulation. The practical weakness is the large number of inequalities that have to be introduced. Techniques like ordering the variables and taking intelligent subsets of all possible inequalities reduce its number [Pug06].

Dynamic symmetry breaking inequalities introduces inequalities during the solution process. Those inequalities might be not correct, when adding them in the beginning. However, at the node in the enumeration tree in which they are introduced it is guaranteed that they do not prevent the discovery of the optimal solution. A well-known dynamic approach is orbit branching [OLRS11, OLRS07]. Orbits are sets of variables that are equivalent with respect to the symmetry remaining in the problem after branching. This can include symmetries that were not existent at the root node. These orbits are then used to create a valid partitioning of the feasible region. This partitioning significantly reduces the effects of symmetry while still allowing a flexible branching rule.

Our cutting plane aggregation approach makes symmetries more explicit, because the interchangeability of variables is more obvious. Without applying our CPA algorithm, symmetry detection algorithms have to detect exchangeable variables among different constraints, taking the weights in the objective into account. To the best of our knowledge, this is beyond the implemented symmetry detection mechanisms of state-of-the-art ILP solvers.

If we apply our CPA method, we formulate a lower number of counting constraints into ILP constraints. The ILP constraints contain more variables within a single constraint all being linked to only one variable in the target function. This makes it easier to check for exchangeable variables. Let us recall Example 12. In this example, every pair of variables y_i is exchangeable. Before aggregation, we had 100 single constraints, each linked with one variable in the target function. Thus, an algorithm would have needed to detect that whole constraints are symmetric, which also includes checking equality of the weights in the objective. After applying the CPA method just one constraint remains, in which algorithms can easily detect that every pair of variables y_i is symmetric.

3.4.2 Computation of Counting Features

Next, special attention is paid on the strategy finding the ground clauses c (see Definition 1) which minimize the number of counting features. This problem can be solved optimally with algorithms that detect symmetries in propositional formulas such as SAUCY [DSM08] or, alternatively, by reducing it to the frequent itemset mining problem for which several robust algorithms as APRIORI [BK02] exist. However, exhaustive experiments showed that time savings due to aggregation were neutralized by executing the exact approach and parsing the received symmetries. Especially, when we combine the CPA with the CPI method (refer to Section 3.5.1) these algorithms need to be called in each CPI iteration.

There's an analogy to selectivity analysis for query processing in relational databases. Since it is infeasible to store the data distribution over all tables, rela-

g	ℓ_1	ℓ_2	ℓ_3	w
g_1	$x_1 \vee$	$\neg y_1 \vee$	z_2	1.0
g_2	$x_2 \vee$	$\neg y_1 \vee$	z_2	1.0
g_3	$x_1 \vee$	$\neg y_1 \vee$	z_2	1.0
g_4	$x_3 \vee$	$\neg y_1 \vee$	z_3	1.0
g_5	$x_1 \vee$	$\neg y_1 \vee$	z_3	1.0
g_6	$x_4 \vee$	$\neg y_1$		1.0

Table 3.5: Example clauses for the approximate counting feature algorithm. For $k = 1$ we result in 3 counting features, for $k = 2$ we obtain 5 counting features, and for $k = 3$ we get 4 counting features. Thus, we choose $k^* = 1$ as optimal aggregation strategy.

tional database management systems estimate the best join order by maintaining histograms over columns [Cha98]. Similarly, computing the most compact CPA feature aggregation is inefficient.

Therefore, we implemented a greedy algorithm that only *estimates* the optimal aggregation scheme. This algorithm initially gets a set of ground clauses $\mathcal{G}^f$ from a first-order formula f. Let n be the number of literals of formula f. Each ground clause in $\mathcal{G}^f$ has the form $g = \ell_1 \vee \ldots \vee \ell_m, m \leq n$. All clauses share the same weight because they all originate from one first order formula. Please note that the length m of each ground clause g might be lower than n due to elimination of literals because of evidence.

The algorithm stores, for each first-order clause, the violated groundings g in a table with n columns where each column represents one literal position of the clause. For each column k, we compute the set of *distinct* rows R_k of the table that results from the projection onto the columns $\{1, ..., n\} \setminus \{k\}$. Let $k^* = \operatorname{argmin}_k\{|R_k|\}$ be the minimal number of distinct entries of R_k for all k. The groundings are then aggregated with respect to the rows in R_{k^*}.

Example 13 illustrates how the algorithm works.

Example 13. *Table 3.5 depict 6 ground clauses which are stored in the set $\mathcal{G}^f$. The maximal length of clauses is $n = 3$.*

Now we start the algorithm with $k = 1$. We compute the number of counting features c leaving out ℓ_1. We result in 3 counting features:

$$\neg y_1 \vee z_2, \quad \neg y_1 \vee z_3, \quad \neg y_1$$

For $k = 2$, we result in 5 counting features:

$$x_1 \vee z_2, \quad x_2 \vee z_2, \quad x_3 \vee z_3, \quad x_1 \vee z_3, \quad x_4$$

Setting $k = 3$ results in 4 counting features:

$$x_1 \vee \neg y_1 \quad x_2 \vee \neg y_1 \quad x_3 \vee \neg y_1 \quad x_4 \vee \neg y_1$$

Finally, we choose $k^* = 1$ as optimal aggregation strategy because it results in the lowest number of counting features.

3.5 Combining and Parallelizing the Cutting Plane Aggregation and the Cutting Plane Inference Appraoch

Within this section we discuss how we can combine the CPA algorithm with the CPI concept and explain how to parallelize this combined algorithm.

3.5.1 Combining the CPI and the CPA Approach

Combining the cutting plane aggregation algorithm, which we discussed in Section 3.4, and the cutting plane inference concept, which was introduced in Section 3.3, is relatively straight-forward. Intuitively, we have to apply cutting plane aggregation on the violated constraints, which we get in each cutting plane inference loop. The modified algorithm is depicted in Algorithm 2.

Algorithm 2 Algorithm which integrates CPA into CPI.

Input: $\mathcal{G}$: Set of all ground clauses
Input: **E**: Set of all evidence axioms
Output: H^*: Maximum a-posteriori state
PERFORMCPIANDCPA

1: $\mathcal{G}_{\mathsf{ILP}} \leftarrow \mathbf{E} \cup$ all $g \in \mathcal{G}$ having one literal only and a weight $w_g > 0$.
2: Initial solution $H^{(0)} \leftarrow$ all atoms in $\mathcal{G}_{\mathsf{ILP}}$.
3: transfer each $g \in \mathcal{G}_{\mathsf{ILP}}$ to linear constraints and add them to the ILP.
4: $t \leftarrow 0$
5: **repeat**
6: $\quad \mathcal{G}_{\mathsf{new}} \leftarrow \varnothing$
7: $\quad$ **for every** ground clause $g \in \mathcal{G} \setminus \mathcal{G}_{\mathsf{ILP}}$ **do**
8: $\quad\quad$ **if** (weight $w_g > 0$ or $w_g = \infty$ **and** g is **not** satisfied with $H^{(t)}$) **or** (weight $w_g < 0$ **and** g is satisfied with $H^{(t)}$) **then**
9: $\quad\quad\quad$ add g to $\mathcal{G}_{\mathsf{new}}$.
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: $\quad$ **if** $\mathcal{G}_{\mathsf{new}} \neq \varnothing$ **then**
13: $\quad\quad t \leftarrow t + 1$
14: $\quad\quad \mathcal{G}_{agg} \leftarrow$ *aggregation of set* $\mathcal{G}_{new}$ *according to Lemma 2.*
15: $\quad\quad$ *transfer each* $G \in \mathcal{G}_{agg}$ *to linear constraints and add them to the ILP.*
16: $\quad\quad$ add every $g \in \mathcal{G}_{\mathsf{new}}$ to the set $\mathcal{G}_{\mathsf{ILP}}$.
17: $\quad\quad$ solve the ILP and set $H^{(t)}$ to the solution of the ILP.
18: $\quad$ **end if**
19: **until** $\mathcal{G}_{\mathsf{new}} = \varnothing$
20: **return** $H^{(t)}$

This algorithm differs only in Line 14 and Line 15 from Algorithm 1. While we originally just translated the new ground atoms $\mathcal{G}_{\mathsf{new}}$ to ILP constraints, we now

apply the cutting plane aggregation algorithm on the set $\mathcal{G}_{\mathsf{new}}$ (Line 14) and add the aggregated clauses to the ILP (Line 15).

Example 14 illustrates the benefit of combining the cutting plane inference with the cutting plane aggregation approach. In the example, we show that applying the concept of cutting plane aggregation together with the cutting plane inference algorithm compared to just applying the cutting plane inference algorithm reduces the constraints from 10 to 2. The example is kept simple as it aggregates clauses with just one literal. For an example, in which the CPA approach aggregates more than one constraint, we forward to Section 7.1. In our experiments in Chapter 6.4.1, we will show that our CPA method also aggregates many formulas with more than one literal.

Example 14. *Let us revisit the example MLN in Table 3 and consider the clause* $\neg\mathsf{smokes}(x) \vee \neg\mathsf{friends}(x,y) \vee \mathsf{smokes}(y)$ *with weight* $w = 1.1$. *Let us assume that there are* 100 *constants* $C_1, ..., C_{100}$ *for which we have evidence that the first 20 constants are friends with each other:* $\mathsf{friends}(C_i, C_j), 1 \leq i, j \leq 20$. *Furthermore, we know that the first 10 constants smoke:* $\mathsf{smoke}(C_i), 1 \leq i \leq 10$.

If we apply neither the cutting plane inference nor cutting plane aggregation approach, the resulting ILP consists of $100 \cdot 100 = 10,000$ *constraints.*

If we only apply the cutting plane inference concept, we result in 100 *violated ground clauses (before applying evidence)*

$$\neg\mathsf{smokes}(C_i) \vee \neg\mathsf{friends}(C_i, C_j) \vee \mathsf{smokes}(C_i), \quad 1 \leq i \leq 10 \leq j \leq 20$$

which are reduced to 10 *ground clauses due to given evidence*

$$\mathsf{smokes}(C_i), \quad 10 \leq j \leq 20.$$

Thus, the resulting ILP would consist of 10 *restrictions. The temporary solution after the first iteration is* $\mathsf{smokes}(C_i)$ *for* $1 \leq i \leq 20$. *Since no more constraints are violated, this temporary solution is also the final MAP state which we reached in* 1 *iteration.*

If we now combine the CPI approach with the CPA method, we can aggregate the 10 *ground clauses with* $c = false$ *and* $\ell_i = \mathsf{cancer}(C_i), 10 \leq i \leq 20$. *This results in only* 2 *ILP constraints*

$$\sum_{i=10}^{20} \mathsf{cancer}(C_i) \geq z_g$$

$$z_g \leq 10$$

where z_g *is an integer variable. The objective just consists of* $\max 1.1 z_g$.

3.5.2 Parallelizing the CPI and the CPA Approach

We are able to parallelize important parts of the MAP inference algorithm. While there are some generic parallel machine learning architectures such as GRAPHLAB

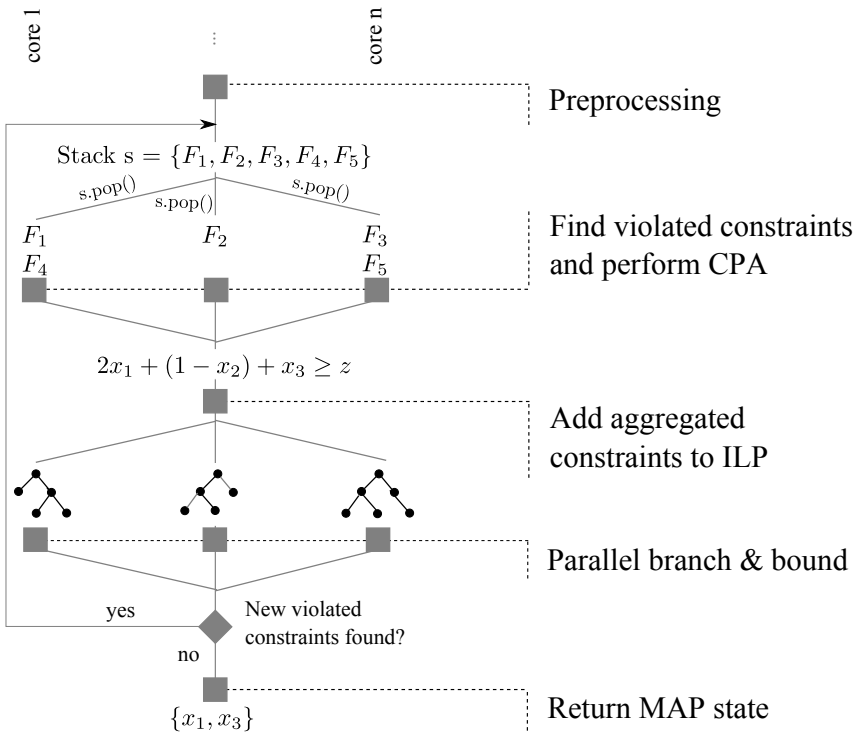

Figure 3.1: Parallelization of constraint finding, constraint aggregation, and ILP solving [NNS13].

[LGK+10], which could in principle be used for parallel MAP inference, we provide the first system that parallelizes MAP inference in statistical relational models combining the CPI and the CPA method. Our parallelization algorithm is summarized in Figure 3.1 and will now be explained in more detail.

First, the algorithm reads in the Markov logic network, which includes the weighted first-order formulas and the evidence axioms. Based on this information, it creates relational database tables and fills them with values. We refer the reader to Chapter 4 for further details.

After preprocessing, we perform several but at least one CPI rounds until no new violated constraints are found. Within one CPI round, the algorithm first finds violated ground clauses and performs the CPA algorithm on them. This process is parallelized since each first order formula can be processed separately. Therefore, we initially put each first order formula in a stack s. Then, we create one thread per available processor. Each of these threads now fetches a first order formula from s, processes it, and fetches the next formula until s is empty. Please note, that for the first round, we can skip the process of finding violated constraints. Instead, we encode all evidence axioms and all ground clauses with positive weights which consist of one literal only to the ILP. For further information about the realization of finding the violated constraints within each CPI iteration and the computation

of the counting features for the CPA approach we forward the reader to Section 4.2 and Section 4.3.

Afterwards, all aggregated clauses from each formula are fetched from each thread and are combined. Then, they are transferred to ILP constraints and added to the ILP from the previous iteration. Please note that we reuse the ILP from the previous iteration instead of creating a new one due to efficiency reasons.

Finally, we compute the (temporary) MAP-state utilizing an ILP solver which is able to solve mixed integer problems in parallel. The basic idea of parallelizing branch-and-bound algorithms is based on the fact that different nodes in the MIP tree search can be processed independently. In the following, we will briefly sketch a standard parallelization algorithm [BBG88]. Initially, they create one process per available processor and assume that all those processes are free. Every free process fetches the next node with the maximum upper bound and calculates its solution. If this solution is integer but not the optimal solution, its objective is stored centrally as the current best lower bound. If the solution is not integer, it introduces a new cut and creates new nodes which need to be processed. If all nodes are processed the problem is solved. Thus, the nodes within the search tree are computed in parallel. Generally, parallelization works better for models, which have to explore a large search tree while models which spend most of the time solving the root node can not be efficiently parallelized.

Chapter 4

Leveraging Relational Database Management Systems (RDBMS)

Within this chapter, we precisely show how we can leverage relational database management systems (RDBMS) to perform important tasks like grounding (refer to Section 4.1), finding violated constraints (refer to Section 4.2) for each CPI loop, and performing the feature aggregation within the CPA approach. In this way, we take advantage of the highly sophisticated query optimization and indexing technologies of relational database systems. Thus, this chapter exhaustively answers the second part of research question Q2 of Section 1.2.1:

Q2 *How can we parallelize the solution process and tightly integrate relational database management systems (RDBMS) within this process?*

The general idea of exploiting RDBMS for CPI was first proposed by Riedel [Rie08]. However, the construction of the queries that are necessary to receive the violated ground clauses was only briefly sketched in his work. In this section, we provide detailed algorithms how the query construction is performed. Furthermore, we illustrate strategies how the counting features of our CPA approach can be implemented with RDBMS 4.3.

Please note that we decided to use the SQL syntax rather than relational algebra operators throughout this chapter since we assume that readers feel more comfortable reading SQL queries. However, the translation of the SQL query construction algorithms to algebra operators is straight forward. We refer the reader to [Bea09, CO93, Dat89, Bra84] for further references to SQL, relational algebra, and their interconnection.

All sections in this chapter are our work. The translation to SQL queries has been very briefly sketched in our work [NN11, NNS13].

4.1 Grounding

Grounding of Markov logic networks can be efficiently implemented in RDBMS. This section describes how SQL queries are constructed to compute all groundings efficiently. We formulate the SQL queries in a way that query optimization techniques of state-of-the-art RDMS systems can greatly improve the execution speed by optimizing the join order. The underlying theory for this section is discussed in Section 2.3.2.

As a prerequisite, we first have to encode evidence of observed predicates in relational database tables. Evidence of hidden predicates is filtered as a post-processing step afterwards. Furthermore, evidence of hidden axioms are implicitly considered when finding the violated constraints for cutting plane inference. We refer the reader to the next section for details.

For every observed predicate p_o/n with arity n, we create a table P_o with n columns. The name of the table equals the name of the observed predicate. We name the columns `field0`, ..., `field(n-1)`. All atoms $p_o(t_1, ..., t_n)$ for the observed predicate p_o which are $true$ due to evidence are stored in one row each. Each term t_i is stored in a separate cell in the ith column. Thus, all atoms that are not stored in table P_o are $false$ due to given evidence.

Then, we construct one SQL query per first-order formula f. For a first-order formula f let $L_o^+(f)$ be the set of observed atoms occurring unnegated in f and $L_o^-(f)$ be the set of observed atoms occurring negated in f. Furthermore, we define sets $\mathcal{S}_v$ for each variable v occurring in f. Initially, those sets are empty. During the following algorithm, these sets will be used to store the position of variable v within predicates.

Algorithm 3 constructs the SQL statement for one first order formula f. The SQL query performs an inner join for every negated atom (line 9) and subtracts the evidence for every positive atom (line 11 and 12). Since subtracting tables is not a standard feature in SQL, we perform a left join with the table P_o and keep only those values of P_o which are `null` [CO93]. On a first glimpse, the reader might think this should be the other way round. However, we implicitly transform the formula f from

$$\bigvee \ell^o \bigvee \ell^h$$

where ℓ^o represents an observed literal and ℓ^h a hidden literal to the form

$$\bigwedge \neg \ell^o \Rightarrow \bigvee \ell^h.$$

Thus, we have a conjunction of all negated observed literals which result in the described join strategy.

The algorithm needs at last one negated atom to be able to build the `FROM` clause (line 7). More precisely, it requires a negated atom for every variable v which only occurs in non-negated observed atoms. For every such variable v, we thus add a dummy literal $\neg p(v)$ to f in a pre-processing step. The assigned type of variable v in the new literal $\neg p(v)$ is inherited. Note that if v is typed, the set of

Algorithm 3 SQL query for computing the grounding of formula f.

Input: f: a first order clause
Input: $L_o^+(f)$, $L_o^-(f)$: set of observed atoms occurring negated / unnegated in f
Output: SQL a SQL statement for grounding formula f

GETSQLFORGROUNDING

$\mathcal{S}_v \leftarrow \varnothing$ for every variable v in f.
FROM $\leftarrow$ `''`
WHERE $\leftarrow$ `'WHERE '`
$i \leftarrow 0$
for every atom p_o in $L_o^-(f) \cup L_o^+(f)$ with variables $v_1, ..., v_n$ **do**
  if atom p_o in $L_o^-(f)$ **and** $i = 0$ **then**
    FROM $\leftarrow$ FROM + `' FROM` P_o `x0 '`
  else if atom p_o in $L_o^-(f)$ **and** $i > 0$ **then**
    FROM $\leftarrow$ FROM + `' INNER JOIN` P_o `x`i `ON '`
  else if atom p_o in $L_o^+(f)$ **then**
    FROM $\leftarrow$ FROM + `' LEFT JOIN` P_o `x`i `ON '`
    WHERE $\leftarrow$ WHERE + `'x`i`.field0 IS NULL' [+ 'AND']`
  end if
  for every variable $v_j \in \{v_1, ..., v_n\}$ **do**
    for every element s in $\mathcal{S}_{v_j}$ **do**
      FROM $\leftarrow$ FROM + `'x`i`.field`j `= `s` ' [+ 'AND']`
    end for
    add `'x`i`.field`j`'` to set $\mathcal{S}_{v_j}$
  end for
  $i \leftarrow i + 1$
end for
SELECT $\leftarrow$ 'SELECT '
for every variable v in f **do**
  SELECT $\leftarrow$ SELECT + `'`s` ' [+ ', ']` where $s \in \mathcal{S}_v$
end for
SQL $\leftarrow$ SELECT + FROM + WHERE

substitutable constants of v may be smaller than the Herbrand universe. We refer to Section 2.3 for a definition of types.

In the second part of the algorithm (line 14-19) we construct the join predicates. Intuitively, we introduce a join predicate for every pair of equal variables. This is done by storing the table and field ids of each prior occurrence of variable v_j in a set $\mathcal{S}_{v_j}$. If we find a new occurrence of variable v_j we add a join predicate for every entry in $\mathcal{S}_{v_j}$ (line 15-17).

In this way, the SQL query contains a the maximal number of join predicates which provides the relational database management system with maximal information about the interlinking between joins. This information is efficiently used by RDBMS for optimizing the join order.

The `SELECT` part includes every variable v of formula f. For its construction we can take a random element s from $\mathcal{S}_v$ as representation for variable v (line 23-24). In the end, we put the constructed query parts together to the final SQL query (line 25).

For the sake of clarity, we used the notation `[+ 'AND']` twice in the algorithm. It means that the string `'AND'` is only attached if we did not reach the last element of the list. This prevents that the `where` statement (line 12) and the join predicates (line 16) end with the command `'AND'`, respectively. The notation `[+ ', ']` works analogous. Please note, that for reasons of readability Algorithm 3 does not support constants within literals like $friend(\text{“}Anna\text{”}, x)$. However, the required extension of the algorithm is trivial, since we just have to include them in the `where` clause.

Example 15 reuses the *smokers* example to explain the above definitions. The SQL query is relatively simple, since we only have one observed predicate in the example formula. In the next section, we will extend the current query formulation to find violated constraints within a cutting plane inference loop.

Example 15. *Let us revisit the friends & smokers MLN from Example 3. Let us define the predicate* friends *as observed predicate and* smokes *as hidden predicate. Let us assume we have given the evidence:* $\mathsf{friends}(\mathsf{Anna}, \mathsf{Bob}), \mathsf{friends}(\mathsf{Bob}, \mathsf{Anna})$ *Then, we initially create the following table:*

friends	
field0	*field1*
Anna	Bob
Bob	Anna

Then, the SQL query to ground the formula

$$\neg\mathsf{smokes}(x) \vee \neg\mathsf{friends}(x, y) \vee \mathsf{smokes}(y)$$

is the following

```
SELECT x0.field0 as x, x0.field1 as y FROM friends x0
```

and has the result

x	y
Anna	Bob
Bob	Anna

which translates to the ground clauses

$$\neg\mathsf{smokes}(\mathsf{Anna}) \vee \mathsf{smokes}(\mathsf{Bob}) \quad \textit{and} \quad \neg\mathsf{smokes}(\mathsf{Bob}) \vee \mathsf{smokes}(\mathsf{Anna})$$

4.2 Finding Violated Constraints

In the previous section, we showed how RDBMS can be efficiently used for grounding Markov logic networks. Now, we get one step further and compute the violated constraints required for cutting plane inference. We refer to Section 3.3 for a detailed discussion of cutting plane inference. For this purpose, we will extend the SQL formulation of the grounding in a way that it returns only the violated constraints.

First, we need to create additional tables in which we store the temporary solution of the last CPI iteration. As we did for the observed predicates before, we now create a table P_h with n columns for every hidden predicate p_h/n with arity n. The name of the table equals the name of the hidden predicate. Again, we name the columns `field0`, ..., `field(n-1)`. Let $H^{(t)}$ be the current intermediate solution of iteration t. Then, we store all atoms $p_h(t_1, ..., t_n)$ of all predicates which are $true$ in the intermediate solution $H^{(t)}$ in the respective table P_h. Each term t_i is stored in a separate cell in the ith column. Thus, all atoms which are not stored in table P_h are $false$ in the intermediate solution $H^{(t)}$. Please note, that the intermediate solution also contains the evidence axioms for the hidden predicates, which are thus implicitly included in the query.

As a prerequisite, we need to define the negated and non-negated set of hidden atoms. For a first-order formula f let $L_h^+(f)$ be the set of hidden atoms occurring unnegated in f and $L_h^-(f)$ be the set of hidden atoms occurring negated in f.

In our SQL query, we want to query the set $\mathcal{G}_{\mathsf{new}}$ which contains only the *new* violated constraints of the current CPI iteration. Thus, we need to subtract the violated constraints $\mathcal{G}_{\mathsf{ILP}}$ from the previous rounds. This is done by introducing a table F for every first order formula f. The columns v_1,...,v_n of table F are representing all distinct variables of formula f.

We first discuss how to retrieve violated constraints if the first order formula has a positive weight $w > 0$ and afterwards we turn our attention to the more complex case of negative weights $w < 0$. Finally, we examine the case of positive and negative weights of conjunctions.

Clauses With Positive Weights

If the clause $f = \ell_1 \vee \ldots \vee \ell_n$ with hidden literals ℓ_i has a positive weight $w > 0$ all violated ground clauses are defined as the set of all ground clauses where $\neg f =$

Algorithm 4 Algorithm for generating the SQL query for finding the violated constraints of a first order formula with positive weights.

Input: f: a first order formula
Input: $L_o^+(f)$, $L_o^-(f)$: set of observed atoms occurring negated / unnegated in f
Input: $L_h^+(f)$, $L_h^-(f)$: set of hidden atoms occurring negated / unnegated in f
Output: SQL a SQL statement for grounding formula f

GETSQLFORCPIPOSWEIGHT

execute GETSQLFORGROUNDING(f, $L_o^+(f)$, $L_o^-(f)$). Keep all internal variables initialized.
for every atom p_h in $L_h^-(f) \cup L_h^+(f)$ with variables $v_1, ..., v_n$ **do**
  if atom p_h in $L_h^-(f)$ **and** $i > 0$ **then**
    FROM ← FROM + ` INNER JOIN P_h xi ON '
  else if atom p_h in $L_h^+(f)$ **then**
    FROM ← FROM + ` LEFT JOIN P_h xi ON '
    WHERE ← WHERE + `x$i$.field0 IS NULL' + `AND'
  end if
  for every variable $v_j \in \{v_1, ..., v_n\}$ **do**
    for every element s in $\mathcal{S}_{v_j}$ **do**
      FROM ← FROM + `x$i$.field$j$ = $s$ ' [+ `AND']
    end for
    add `xi.field$j$' to set $\mathcal{S}_{v_j}$
  end for
  $i \leftarrow i + 1$
end for
FROM ← FROM + ` LEFT JOIN F f ON '
WHERE ← WHERE + `f.v IS NULL' where v is any variable in f
for every variable v in f **do**
  for every element s in $\mathcal{S}_v$ **do**
    FROM ← FROM + `f.$v$ = $s$ ' [+ `AND']
  end for
end for
SQL ← SELECT + FROM + WHERE

$\neg\ell_1 \wedge \ldots \wedge \neg\ell_n$ holds in the current intermediate solution $H^{(t)}$. Since $\neg f$ consists of conjunctions of literals, we can utilize SQL joins with the tables in which the intermediate solution is stored to retrieve all ground clauses.

Algorithm 4 constructs the SQL query for computing the violated constraints. It is directly based on Algorithm 3 and extends the computed SQL snippets SELECT, FROM, and WHERE. Thus, we take the final strings of those SQL snippets from Algorithm 3 as initial values for Algorithm 4 (line 1). Furthermore, we also take the final content of the sets $\mathcal{S}_v$ for each variable v and the value of variable i of Algorithm 3 as input for Algorithm 4.

For retrieving every violated constraints we implicitly transfer the formula f to $\neg f$, turning the disjunction of literals to conjunction of the respective negated literals. Thus, in case of a (originally) negated literal p_h we perform an inner join on its intermediate solution table P_h (line 4). In case of a non-negated literal, we subtract its respective table P_h by performing a left join on P_h and keep only those values of P_h which are `null` (line 6-7).

In the second part of the algorithm (line 9-14) we construct the join predicates. This is exactly the same as in Algorithm 3. Intuitively, we introduce a join predicate for every pair of equal variables. This is done by storing the table and field ids of each prior occurrence of variable v_j in a set $\mathcal{S}_{v_j}$. Then, if we find a new occurrence of variable v_j we add a join predicate for every entry in $\mathcal{S}_{v_j}$ (line 10-12).

The third part of the algorithm (line 17-23) ensures that we only get the *new* violated constraints. Thus, we subtract table F which contains the violated constraints of formula f found in previous iteration (line 17-18). In line 19-23 we then build the joint predicates for table F analogous to line 9-14.

As in Algorithm 3, we used the notation `[+ 'AND']` in line 11 and line 21 meaning that the join predicate items are connected with the string `'AND'` but must not end with it.

Example 16 illustrates Algorithm 4 and its requirements. After query execution, the results are then directly used to build ILP constraints and are stored in the respective table F for duplicate detection in the next iteration.

Example 16. *Let us revisit Example 15 from the last section. We are interested in the SQL query for finding the violated constraints for the first-order formula*

$$f1 := \neg\mathsf{smokes}(x) \vee \neg\mathsf{friends}(x, y) \vee \mathsf{smokes}(y).$$

Additionally to the existing `friends` *table, we now construct a* `smokes` *table for the hidden predicate* smokes *with one column* `field0`*. We assume that the intermediate solution* $H^{(0)}$ *in iteration* $t = 0$ *contains* smokes(Anna)*, which is implicitly the evidence for the hidden predicate* smokes*. The evidence is put into the* `smokes` *table. Furthermore, we create a table* `f1` *storing all previously violated constraints for duplicate detection. Since we are in the first iteration, table* `f1` *has no entries yet. In summary, we now have the following tables:*

friends	
field0	field1
Anna	Bob
Bob	Anna

smokes
field0
Anna

f1	
x	y

Then, the SQL query for finding the violated constraints of formula $f1$ *is the following.*

```
SELECT x0.field0 as x, x0.field1 as y FROM friends x0
INNER JOIN smokes x1 ON x1.field0 = x0.field0
LEFT JOIN smokes x2 ON x2.field0 = x0.field1
LEFT JOIN f1 f ON f.y = x0.field1 AND f.x = x0.field0
AND f.x = x1.field0
WHERE x2.field0 IS NULL AND f.y IS NULL
```

and has the result

x	y
Anna	Bob

which translates to the ground clause

$$\neg\mathsf{smokes}(\mathsf{Anna}) \vee \mathsf{smokes}(\mathsf{Bob})$$

Algorithm 5 Algorithm for generating the SQL query for finding the violated constraints of a first order formula with negative weights.

Input: f: a first order clause
Output: SQL a SQL statement for grounding formula f

GETSQLFORCPINEGWEIGHT

1: $\widehat{f} \leftarrow$ formula consisting of all observed literals of f.
2: $\ell_{old} \leftarrow$ NULL
3: SQL $\leftarrow$ ''
4: **for every** hidden literal ℓ in f **do**
5: **if** $\ell_{old} \neq$ NULL **then** negate ℓ_{old} in $\widehat{f}$ **end if**
6: Add ℓ to $\widehat{f}$.
7: $\tilde{f} \leftarrow \widehat{f}$ where each hidden literal ℓ is negated.
8: SQL $\leftarrow$ SQL + GETSQLFORCPIPOSWEIGHT($\tilde{f}$) [+ ' UNION ALL ']
9: $\ell_{old} \leftarrow \ell$
10: **end for**

Clauses With Negative Weights

In case of *negative* weights $w < 0$, the query for retrieving all violated constraints is more complex. Assume that we have a first order clause f with weight $w < 0$ of the form

$$f := \ell_1 \vee \ldots \vee \ell_n$$

f			$\widehat{f}$
ℓ_1	ℓ_2	ℓ_3	ℓ_1
		$\neg\ell_3$	
	$\neg\ell_2$	ℓ_3	
		$\neg\ell_3$	
$\neg\ell_1$	ℓ_2	ℓ_3	$\neg\ell_1 \wedge \ell_2$
		$\neg\ell_3$	
	$\neg\ell_2$	ℓ_3	$\neg\ell_1 \wedge \neg\ell_2 \wedge \ell_3$
		$\neg\ell_3$	not violated

Table 4.1: An example for finding violated constraints of a first order formula $f := \ell_1 \vee \ell_2 \vee \ell_3$ with negative weight.

with either non-negated or negated hidden literals ℓ_i. For each cutting plane inference iteration we then have to include every ground clause *except* those for which $\neg f = \neg\ell_1 \wedge \ldots \wedge \neg\ell_n$ holds in the current intermediate solution $H^{(t)}$.

The SQL query to retrieve all ground clauses but those for which $\neg f$ holds in the current intermediate solution is a union of several join queries. Those join queries can be constructed with Algorithm 4. Let f be the first order formula with negative weight and let $\widehat{f}$ be a first order formula which only consists of the observed literals of f. Then, the query is constructed as described in Algorithm 5.

The algorithm can be explained best on a generic example. Let us assume we have the first-order formula $f := \ell_1 \vee \ell_2 \vee \ell_3$ consisting of three *hidden* literals ℓ_i. We now apply Algorithm 5 on f. Since this formula has 3 literals, the loop (line 4-10) is executed three times. In the first iteration we obtain $\widehat{f} = \ell_1$. Thus, Algorithm 4 is started with $\tilde{f} = \neg\ell_1$ (line 8). We have to negate $\widehat{f}$ in line 7 because Algorithm 4 computes the SQL query for the negated formula. In the second iteration, we receive $\widehat{f} = \neg\ell_1 \wedge \ell_2$ and in the last iteration we get $\widehat{f} = \neg\ell_1 \wedge \neg\ell_2 \wedge \ell_3$. If we union the formulas $\widehat{f}$ over all three iterations, we obtain every grounding except for $\neg f = \neg\ell_1 \wedge \neg\ell_2 \wedge \neg\ell_3$ which is exactly what we intended. Table 4.1 shows the different sub-formulas $\widehat{f}$ which are generated with respect to its coverage of f.

Conjunctions

Finding the violated constraints for weighted conjunctions of the form $\ell_1 \wedge \ldots \wedge \ell_n$ is based on the algorithms for clauses.

If the weight of the conjunction is positive $w > 0$, the violated constraints are determined with $\neg(\ell_1 \wedge \ldots \wedge \ell_n) = \neg\ell_1 \vee \ldots \vee \neg\ell_n$. Thus, Algorithm 5 can be reused with input formula $f = \neg\ell_1 \vee \ldots \vee \neg\ell_n$.

In case of negative weights $w < 0$, we have to add an ILP constraint within a CPI iteration, if the current solution entails $\ell_1 \wedge \ldots \wedge \ell_n$. Thus, we can reuse Algorithm 4 for clauses with positive weights.

We can conclude that performing cutting plane inference for clauses with positive weights is usually more efficient than for to clauses with negative weights since the number of violated ground clauses is often larger for the latter case. The oppo-

	weight	
	$w > 0$	$w < 0$
clause	more efficient Algorithm 4	less efficient Algorithm 5
conjunction of literals	less efficient Algorithm 5	more efficient Algorithm 4

Table 4.2: A summary for finding violated constraints with RDBMS. We recapitulate algorithms and efficiency for clauses and conjunction of literals with positive and negative weights.

site holds for weighted conjunctions. Here, conjunctions with negative weights are usually more efficient than conjunctions with positive weights. Table 4.2 summarizes these findings.

4.3 Computation of Counting Features

A key component within our CPA approach is the decision which literals should be aggregated to a counting constraint. In Section 3.4.2 we already discussed the computation of counting features on an abstract level. We argued that exact approaches, which find the optimal aggregation strategy, are not feasible, because their runtime dominates the runtime savings due to the aggregation. Thus, we described a greedy approach which was inspired by join order optimization strategies from relational databases. For readability, we now recapitulate the core component of the greedy algorithm of Section 3.4.2:

"The algorithm stores, for each first-order clause, the violated groundings g in a table with n columns where each column represents one literal position of the clause. For each column k, we compute the set of distinct rows R_k of the table that results from the projection onto the columns $\{1, ..., n\} \setminus \{k\}$. Let $k^* = \text{argmin}_k\{|R_k|\}$ be the minimal number of distinct entries of R_k for all k. The groundings are then aggregated with respect to the rows in R_{k^*}.

An efficient way to determine the number $|R_k|$ of distinct entries R_k is to use a `SELECT COUNT` SQL query. Please note, that we implemented the computation of counting features not with RDBMS. For more details and justification, we refer the interested reader to Section 6.1. Let SQL be the SQL query to compute the (violated) groundings for first order clause f. This query is retrieved from Algorithm 3, Algorithm 4, or Algorithm 5 depending whether the CPI method is applied or not and whether the weight of the formula is negative or positive. We now delete the `SELECT` part of the query so that SQL starts with the string `FROM`. Then, we retrieve the number of distinct entries $|R_k|$ with the following query

```
'SELECT COUNT(*) FROM (' +
'SELECT DISTINCT {ν(ℓ_i)|i ∈ {1,...,n} \ {k}} ' + SQL + ')'
```

where the ℓ_is are the hidden literals occurring in the first-order clause f. Literal ℓ_k is not selected. Method $\nu(\ell_i)$ returns a reference to a column for all variables oc-

curring in ℓ_i. We chose a nested query, since for several RMDBS implementations a nested `COUNT` query shows a higher performance than its non-nested alternative. Example 17 constructs a sample query. We now construct queries to compute every $|R_k|$ and determine the optimal d with $d = \text{argmin}_k\{|R_k|\}$.

Example 17. *We recapitulate Example 16. Within this example we presented the SQL query for retrieving the violated constraints of formula*

$$f1 := \neg\mathsf{smokes}(x) \vee \neg\mathsf{friends}(x, y) \vee \mathsf{smokes}(y).$$

We now want to determine the aggregation strategy according to our greedy algorithm. Since the predicate friends *is defined to be observed,* $n = 2$ *hidden literals* $\ell_1 = \mathsf{smokes}(x)$ *and* $\ell_2 = \mathsf{smokes}(y)$ *remain. After removing the current* `SELECT` *part, variable* SQL *has the value*

```
SQL := FROM friends x0
INNER JOIN smokes x1 ON x1.field0 = x0.field0
LEFT JOIN smokes x2 ON x2.field0 = x0.field1
LEFT JOIN f1 f ON f.y = x0.field1 AND f.x = x0.field0
AND f.x = x1.field0
WHERE x2.field0 IS NULL AND f.y IS NULL
```

We now provide some examples that illustrate the return values of method $\nu()$*. We obtain* $\nu(\mathsf{smokes}(x))$ = `x0.field0` *because* `x0.field0` *is a reference to a column of variable* x*. Another possible value for* $\nu(\mathsf{smokes}(x))$ *is* `x1.field0`*. We can choose randomly between those values. Please note that if the literal contains more than one distinct variable, more than one reference is returned.*

Finally, the query for computing the number of distinct entries $|R_2|$ *(including literal* $\neg\mathsf{smokes}(x)$ *and excluding literal* $\mathsf{smokes}(y)$*) is the following:*

```
'SELECT COUNT(*) FROM (SELECT DISTINCT x0.field0 ' + SQL
+ ')'
```

The algorithm can be optimized by storing the result of the original SQL query in a temporary table and query this temporary table with the above `COUNT` queries instead of recomputing it every time. Our experiments showed that when SQL is complex, the execution time of the queries for determining d was reduced when results were pre-computed. However, if the result of the original query was large, we noticed a performance decrease when we pre-stored the result. Overall, runtimes of both strategies were comparable.

Chapter 5

Related Work

Our main contribution in Part I is a new aggregation method for the translation of MAP queries to ILPs. With the help of this translation, we make symmetries explicit to state-of-the-art ILP solvers. Section 5.1 summarizes over several so-called lifted inference approaches, which also detects symmetries in statistical relational languages. However, those approaches require large sets of indistinguishable objects which is usually not given in real-world networks.

In their main Markov logic publication, Richardson and Domingos [RD06] proposed MAXWALKSAT for solving MAP-queries. MAXWALKSAT is an approximate random walk algorithm for solving weighted SAT problems [KSJ97]. Section 5.2 explains the algorithm and its most important extensions in detail.

MAXWALKSAT is the standard inference algorithm for MAP inference implemented in the most established MLN engine ALCHEMY [DJK+]. More recently, the MLN engine TUFFY [NRDS11] used relational database management systems (RMDBS) to ground Markov logic networks more efficiently. TUFFY also runs MAXWALKSAT on the ground model. The MLN system MARKOV THEBEAST basically implements Riedel's CPI inference algorithm [Rie08] presented in Section 3.1 and Section 3.3. Section 5.3 discusses all three systems in more detail.

Section 5.1 has been summarized from many different publications within the area of lifted inference. The initial *hair color* example is taken from [Poo03]. Some text snippets might be identical with the related work section of our publication [NNS13]. However, our summary in this thesis is much more exhaustive. Section 5.2 is a summary originated from many sources including [KSJ97, RD06, NRDS11, SD06b]. The weighted SAT algorithm (Algorithm 6) is a combination of the algorithm presented in [NRDS11, KSJ97] with small corrections. The summary about state-of-the-art Markov logic systems is based on the respective publications of the systems [Rie08, NRDS11], information on the systems' websites, and personal experience gained while conducting the experiments.

5.1 Lifted Inference

Lifted inference performs inference directly on the first-order logic clauses instead of operating on the ground clauses. It exploits symmetries in the probabilistic models resulting from statistical relational languages. The benefit of investigating symmetries on the first-order logic level is that full grounding of Markov logic network can be avoided.

In lifted inference variables are assumed to be random variables over the groundings. An atom $\mathsf{hairColor}(x)$ could for example have the probability of 0.01 for x = purple, meaning that one per-cent of the groundings of x have $\mathsf{hairColor}(purple)$. Thus, lifted inference treats sets of indistinguishable objects as one. If we have many large sets of indistinguishable objects, lifted inference can yield exponential speed-ups over traditional approaches [Poo03].

As the name implies, the main principle of lifted inference is to take an existing probabilistic inference algorithm like the ones we presented in Section 2.1.2 and lift it by carrying out inference over groups of random variables that behave similarly during the algorithm's execution. In other words, these algorithms are often lifted version of standard (probabilistic) inference algorithms [JGMS10].

In the following, we first discuss the more established lifted inference algorithms for marginal inference and then turn to the more related work by summarizing the work dealing with lifted MAP inference. Finally, we will briefly introduce the recent idea of lifting linear programming.

Lifted Marginal Inference The field of lifted marginal inference is better explored than lifted MAP inference. Thus, we first briefly sketch historically important work in this field although lifted MAP inference is more closely related to our novel CPA approach.

The first lifted marginal inference approach is based on first-order variable elimination (FOVE) [Poo03]. They provide an exact inference approach in which they combine first order resolution with variable elimination. Analogous to substitution in theorem proving, they introduce a splitting operation, in which they are not only concerned about the instances created, but also about the instances left over. Milch et al. [MZK$^+$08] further improved the FOVE algorithm by introducing counting formulas to compactly represent interchangeability within large potentials during variable elimination. Another improvement is made by Kirsynski et al. [KP09] who introduced a new data structure called aggregation parfactors.

Singla and Domingos [SD08] provide the first lifted inference algorithm for exact and loopy belief propagation. They create a set of supernodes and superfeatures, corresponding to sets of nodes and features that are indistinguishable given the evidence, and apply belief propagation to this network. Later, Kersting et al. [KAN09] provide counting belief propagation as a generalization of Singla and Domingos lifted belief propagation algorithm. Their approach can be applied to any factor graph without necessarily knowing the corresponding first-order formulas. In a first step they compute a compressed factor graph. This is done by sim-

ulating belief propagation while tracking which nodes and factors send the same messages. If they sent the same messages the corresponding nodes and factors are grouped together. In a second step, they perform belief propagation on the compressed factor graph.

Another branch of literature is lifted knowledge compilation and theorem proving [dB11, dBTM+11, GD12] which is inspired by logical resolution where lifted inference is commonly performed. The algorithm of [GD12] exploits the symmetries within the logical structure of the first order model which allows more computation to be done on the first-order level. This is done by transferring the first-order probabilistic inference task to a first-order weighted model counting problem. In weighted model counting we determine the sum of the weights that satisfies a knowledge base. In their experiments they show that they can identify and lift additional structures which were not discovered by Milch et al. [MZK+08]. In [dBTM+11, dB11] they also exploit model counting for lifted inference and focus on compiling circuits. Furthermore, they showed that some existing lifted inference approaches have a model-theoretic counterpart. Their definition of domain lifted inference as a class of problems in which inference must run in polynomial time helps to characterize the classes of probabilistic models to which lifted inference applies. Gogate and Domingos [GD10] propose a lifted version of AND/OR search which is closely related to [dBTM+11] except that the latter one supports caching and introduced additional operators.

Recently, lifted Markov chain Monte Carlo [Nie12, VG12] approaches arose from classical particle based sampling methods. Niepert [Nie12] constructs a colored undirected graph whose automorphism groups are equivalent to the permutation groups. These permutation groups are then used to build an orbital Markov chain on which sampling is performed. He shows that it converges faster than traditional MCMC sampling in both experiments and theory. Venugopal and Gogates [VG12] propose an algorithm which explicitly considers the structure of the corresponding first-order logic clauses. Their main idea is to partition the first-order atoms in the model into disjoint clusters in a special way. Given such a set of clusters, they transfer the Gibbs sampling to a message passing algorithm. Each message from a sender to a receiving cluster is a truth assignment to all ground atoms that are in the Markov blanket of the receiving cluster.

While there are several other approaches to lifted inference such as bi-simulation based approximate inference [SDG09] all those approaches are limited to lift marginal inference and can not be applied to MAP inference in a straight forward way.

Lifted MAP Inference To the best of our knowledge Braz et al. [dSBAR06] was the first work which explicitly focus on lifted MAP inference. They extended the first-order variable elimination algorithm of Pole [Poo03] by introducing lifted assignments which are assignments over groups of indistinguishable random variables. They distinguish between universally quantified lifted assignments where

the assignment is valid for all values of a set of logical variables and existentially quantified lifted assignments where a specific number of substitutions make a subformula true. Furthermore, they specify a new operator called mpe replacing the maximization function in the non-lifted MAP inference. The mpe operator not only returns the maximum value of the function but also the assignments which lead to this maximum value. Since the mpe operator is decomposable they can then replace the sum operator in their modified FOVE Algorithm [Poo03] with the mpe operator.

Apsel and Brafman [AB12] introduced an approach for MAP inference that takes advantage of uniform assignments. Uniform assignments are groups of random variables that have identical assignments in some intermediate solutions. These uniform assignments can then be replaced by a single representative which simplifies the model so that the MAP state of the compressed model is identical to the MAP state of the original model. In particular, they define new operators which enables to analyze the model's structure regardless of domain size. Their simplification method can be run as preprocessing step. Unfortunately, no code is publicly available.

Automorphism groups of graphical models were used to lift variational approximations of MAP inference [BHR13]. Within a group they summarized random variables and feature functions with identical marginals and expectations. We attempted to compute automorphism groups as an alternative method for aggregating constraints but experiments showed that calling a graph automorphism algorithm in each CPI iteration dominated the overall solving time. The work of Jha et al. [JGMS10] focus on transforming MLNs with a set of transformation rules to simpler ones. However, their approach is restricted to a small subset of rules in which for instance transitivity can not be processed. For example, their approach excludes models that contain a clause where the same predicate appears more than once. Furthermore, it only works if the inference can be done fully on the lifted level.

Lifted Linear Programs Recently, some ideas of lifting linear programs were proposed. Mlandenov et al. [MAK12] computed approximate solutions to linear programs by reducing the LP problem to a pairwise Markov random field over Gaussians and applying lifted Gaussian belief propagation. Their work is build on multi-evidence lifting [AKS11]. In multi-evidence lifting, ground networks for all inference tasks has to be constructed. On the union of these networks a color passing algorithm computes the joint lifted network. With the help of this network symmetries across inference tasks are exploited. Finally, a modified message passing algorithm is run on the joint network. Mlandenov et al. [MAK12] improved the idea of multi-evidence lifting [AKS11] in a way that it can be computed by standard linear program solvers and that it does not have to compute lifted networks in each iteration of the interior-point method again. Similar to the approach of [BHR13] lifted linear programming can be used to approximate LP relaxations [Asa06] of

the MAP ILP.

General Disadvantages and Contribution The main general disadvantage in lifted inference lays in the fact that MLNs from real world problems are often highly connected. They also often contain many evidence axioms and many variables per formula. In other words, the overall number of sets of indistinguishable objects and the sets themselves are very small. In these types of MLNs lifted inference shows no better performance than standard inference approaches.

In many lifted inference papers, they evaluate their approaches with datasets where they artificially populate a relatively simple model with many instances which then results in large sets of indistinguishable objects. Our following experiments are based on real-world datasets where no artificial instances are neither populated nor added.

Contrary to previous work, we use a more compact ILP formulation with a one-to-one correspondence between ground clauses and linear constraints, tightly integrates the CPI and the CPA approach, and *estimates* the optimal aggregation scheme avoiding a costly exact computation in each CPI iteration. Moreover, contrary to lifted inference approaches operating solely on the first-order level, we exploit evidence-induced local symmetries on the ground level.

5.2 Inference in Markov Logic With Weighted SAT Algorithms

The state-of-the art algorithm of solving maximum a-posteriori queries in Markov logic is MAXWALKSAT. The original algorithm [KSJ97] was developed to solve general problems with soft and hard constraints and was proposed for ML by Richardson and Domingos [RD06]. Historically, it was adapted from the WALKSAT [SKC94] algorithm which is one of the fastest and robust algorithms for solving hard formulas in conjunctive normal form. The goal of the algorithm is to maximize the objective of a truth assignment. The objective is calculated by summing up the weight of all satisfied clauses in the current truth assignment σ^*.

The pseudo-code of MAXWALKSAT is displayed in Algorithm 6. As input it needs a set of weighted clauses and a set of atoms occurring in these clauses. Furthermore, the user can specify several parameters including MAXFLIPS, MAXTRIES, and q. The meaning of these parameters are explained below.

The algorithm starts with a random truth assignment to the set of atoms. Beginning with these assignment, it picks one violated ground clause $g \in \mathcal{G}$ randomly. Then, it either flips a random atom in g with probability q or it flips the atom in q which maximizes the objective. Flipping an axiom means to remove it from the truth assignment if contained or add it to the truth assignment if not contained. If the new objective after the flip is larger than the old one, we keep g flipped. Otherwise, we do not flip g and leave everything unchanged. This process is repeated

Algorithm 6 MAXWALKSAT algorithm for solving MAP queries in Markov logic [NRDS11, KSJ97].

Input: $\mathcal{G}$: a set of weighted ground clauses
Input: $\mathcal{A}$: a set of atoms
Input: MAXFLIPS, MAXTRIES, q
Output: σ^* a truth assignment to $\mathcal{A}$

MAXWALKSAT

```
1: OBJGLOBAL ← +∞
2: for 1 to MAXTRIES do
3:    OBJ ← +∞
4:    σ ← a random truth assignment to A
5:    for 1 to MAXFLIPS do
6:       get a random g ∈ G that is violated.
7:       RAND ← a random float number between 0 and 1.
8:       if RAND ≤ q then
9:          flip a random atom a in g and compute the objective OBJTEMP.
10:      else
11:         flip an atom a in g so that the objective OBJTEMP increases most.
12:      end if
13:      if OBJTEMP > OBJ then
14:         OBJ ← OBJTEMP
15:      else
16:         reverse the flip of a.
17:      end if
18:   end for
19:   if OBJ > OBJGLOBAL then σ* ← σ, OBJGLOBAL ← OBJ end if
20: end for
```

MAXFLIPS times. In order to avoid running in local maxima, we execute the algorithm MAXTRIES times, each time starting with a new truth assignment. Finally, the truth assignment with the highest objective is returned.

One of the main drawbacks of the MAXWALKSAT algorithm is that it requires a full grounding of the current Markov logic network. Even for medium sized problems the full grounding can become very large and it's computation is time and memory consuming. LAZYSAT [SD06b] is an optimized version of MAXWALKSAT which only loads ground clauses when they are needed. Instead of computing the total objective, it computes the *change* of the objective when flipping axiom a. Furthermore, it maintains lists containing the active atoms and active ground clauses. Initially, these lists contain all atoms which are not satisfied by the current truth assignment and all clauses activated by those axioms, respectively. An axiom or ground clause is active if it is contained in the the active axioms or active ground clauses list, respectively. By adding an axiom or a ground clause to the corresponding list, we activate it. Every time we flip an axiom a which is not activated we activate a and the clauses in which a occurs. Consequently, there will remain axioms and ground clauses in the end, which were never activated. The number of activated axioms and clauses raises with the number of flips. In their experiments they showed that LAZYSAT reduces the required memory significantly, because a lower number of axioms and clauses needed to be loaded. However, they could not achieve a significant lower runtime compared to MAXWALKSAT. We avoid complete grounding for many real-world problems because we apply cutting plane inference [Rie08].

Riedel [Rie08] employed relational database systems to speed up grounding. Later, Niu et al. [NRDS11] copied this idea and showed in their experiments significant performance and memory savings. Thus, we also employ relational database systems for grounding.

Another drawback of MAXWALKSAT and it's variants is the missing possibility to measure the quality of the result [NRDS11]. Even if it achieved the optimal result, there is no possibility to notice this. Since we exploit ILP solvers we inherit the gap parameter which specifies an approximate solution's worst-case precision.

5.3 State-of-the-art Markov Logic Systems

In this section we will describe existing systems which solve Markov logic networks. In the following comparison, we are especially interested in the systems' algorithms for solving MAP queries. Most of the discussed systems will be used to compare them with our novel Markov logic engine in Section 6.

The first system which was developed for solving Markov logic networks was ALCHEMY[1]. The first version was released in 2006. Very recently, ALCHEMY2[2] has been released. The main difference compared to the previous version is the

[1] http://alchemy.cs.washington.edu/

[2] http://code.google.com/p/alchemy-2/

implementation of lifted inference algorithms. However, the implemented lifted inference algorithms are only applicable for marginal inference. There is no implementation for lifted inference for MAP queries yet. Thus, we will compare our system with ALCHEMY and not ALCHEMY2. For solving MAP queries ALCHEMY and ALCHEMY2 both implement MAXWALKSAT and the more memory efficient LAZYSAT. Those algorithms have been discussed in dept in Section 5.2

Historically, the next MLN system after ALCHEMY was MARKOV THEBEAST[3]. The system was developed between 2007 and 2009 by Riedel [Rie08, Rie09]. It was the first MLN system which exploits databases for grounding. Riedel reimplemented the database language D [DD95, Rie09]. MAP queries are solved with cutting plane inference. We refer to Section 3.3 for an in-dept discussion of the CPI algorithm. Furthermore, it translates ground clauses into ILP constraints as described in Section 3.1. Compared with our translation, Riedel's translation requires more ILP constraints per soft ground clause and does not exploit symmetries within constraints. The original version of MARKOV THEBEAST implements a comparable slow ILP solver. For fair runtime comparisons in the following experimental section, we implemented an interface to GUROBI. GUROBI is currently the fastest available ILP solver and is also used in our novel MLN engine. We forward to Section 6.1 for details.

Recently, the MLN system TUFFY [NRDS11][4] was developed. The newest available version which we will use in our experiments was released in December 2012. They use relational databases for solving MLNs like first intended by Riedel. They also use MAXWALKSAT as MAP inference algorithm. The MLN engines ALCHEMY and TUFFY have a parameter specifying the number of MAXWALKSAT [KSJ97] flips.

ALCHEMY and TUFFY perform several transformations on the set of input formulas so as to obtain clauses with positive weights only [DLK+08, NRDS11]. While we are able to process arbitrary formulas with positive and negative weights under the original Markov logic semantics, we performed the same transformations to achieve comparability to the existing MLN engines[5].

Recently, a general framework for statistical relational learning called KCreator [TFL+10] has been developed which focus on the easy usability and interface designs [KIBFT11]. The MAP inference algorithm uses the Alchemy implementation directly for inference. Since it focuses more on interface designs and integration of many statistical relational languages but not on fast inference algorithms we did not include it in our evaluation.

[3] http://code.google.com/p/thebeast

[4] http://hazy.cs.wisc.edu/hazy/tuffy/download/

[5] The formula transformation can be disabled by setting the parameter `simplify_negative_weight_and_conjunction` to `false` in our MLN engine's configuration file.

Chapter 6

Experiments

With the following experiments we assess the performance and effectiveness of employing our CPA method and parallelization and answer research question Q3:

Q3 *Does our new techniques reduce runtime and outperform existing Markov logic systems with respect to runtime and quality of the results?*

In this context, we compare our novel MLN system ROCKIT with the Markov logic engines MARKOV THEBEAST, TUFFY, and ALCHEMY. We refer to Section 5.3 for a description of the mentioned MLN systems.

To that end, we first describe our Markov logic system ROCKIT in Section 6.1. Then, we turn our attention to the used benchmarks in Section 6.2, discuss the experimental setup in Section 6.3, and present the experimental results in Section 6.4.

Some of the results have already been published in [NNS13], however we performed more exhaustive experiments for this thesis. Some text passages, graphics, and results are copied from our publication [NNS13]. In particular, we show that runtime also decreases when we do not combine the CPA with the CPI method but solely apply CPA without CPI. Furthermore, we indicate that the CPA algorithm also aggregates more complex formulas containing at least two hidden predicates. A shorter version of the technical details of ROCKIT has been published in [NNS13].

6.1 Markov Logic Engine ROCKIT

We have implemented the previously presented concepts of CPA and CPI as well as our algorithms for leveraging relational database management systems in a novel Markov logic engine[1]. In this Chapter we dive a little bit deeper into some of it's components and state and explain our technology choices like the underlying ILP solver and the used database system. In addition to the source-code of ROCKIT, a documentation and installation instructions, we also provide an easy-to-use web-service. The web-service features a web interface where users can upload MLNs

[1] `http://code.google.com/p/rockit/`

and compute MAP states[2]. Figure 6.1 provides a screenshot of the user interface. Furthermore, programmers can integrate the MLN engine in their application via existing REST interfaces.

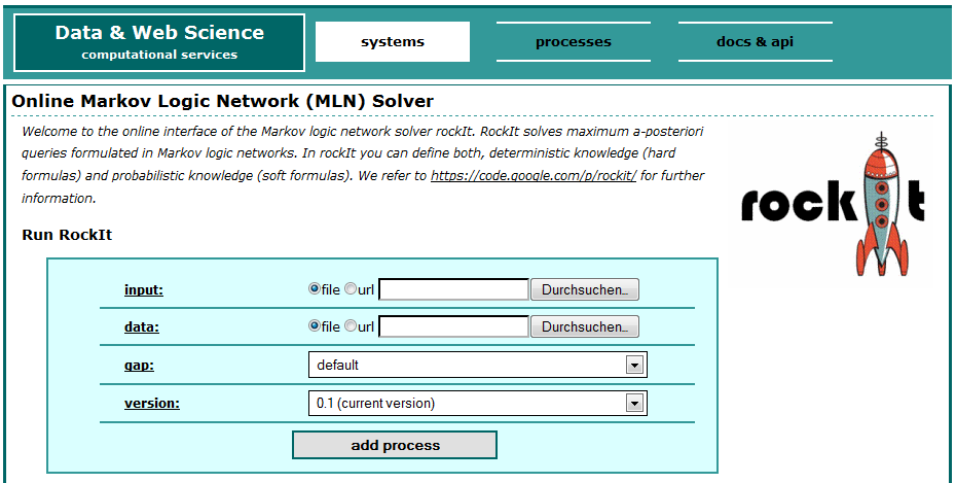

Figure 6.1: Screenshot of the online web-interface of ROCKIT. Users can easily solve their MLNs without any configuration and installation effort.

ROCKIT utilizes the relational database management system MYSQL[3] for efficient computation of groundings and violated constraints for each CPI iteration. We chose MYSQL because of the ability to define in-memory tables. As the name implies, in-memory tables are stored on the main memory. If combined with hash indexes the response times are much lower compared to tables which are stored on hard disk memory.

We achieved further remarkable performance gains by optimizing the procedure of inserting data into the tables. For up to 1000 records we formulate one large `INSERT` statement including every record. For larger number of records, we use the `LOAD DATA` functionality, where MYSQL reads data very efficiently from a comma separated text file. In both techniques, hash indexes are only updated once after all data has been inserted which makes them much faster than single `INSERT` statements. The limit of 1000 records was determined experimentally.

The computation of the counting features was implemented in JAVA and not with SQL queries as intended in Section 3.4.2. To that end, we use hash maps to store all aggregated literals ℓ_k as values and all literals with respect to whom we aggregated all ℓ_k's as keys. During addition we check if some literals can be omitted due to evidence. In the following, we compare the SQL and JAVA variant and motivate our choice towards the JAVA implementation.

[2]The user interface is available at `http://executor.informatik.uni-mannheim.de/systems/rockit/`.

[3]`http://www.mysql.com/products/community/`

The JAVA variant and the SQL variant presented in Section 3.4.2 both use hash technology to determine the k which results in the smallest number of counting constraints. The difference between the variants lies in evidence consideration. In order to explain this, we first have to distinguish between two types of evidence consideration. In type (a) whole clauses can be omitted due to evidence and in type (b) single literals within a clause can be dropped due to evidence. If every literal is dropped, we can omit the whole clause. For more formal definitions we refer to Section 2.3.2. Type (a) is implicitly included in the SQL queries of Algorithm 4 and Algorithm 5. Thus, evidence consideration of type (a) is considered in both the SQL and the Java implementation. However, type (b) is only considered in the Java implementation. There, we can check for evidence *before* we put them in the hash map h_k. In the SQL implementation evidence of type (b) is not considered in the decision of an optimal k since including it in SQL queries leads to complex queries containing if conditions and having longer runtimes. We refer to Section 4.2 for further details.

If evidence of type (b) is not considered it results in slower runtimes because the number of resulting ILP constraints are larger and because the aggregation strategy changes compared to considering evidence of type (b). Consequently, we implemented the Java variant with evidence detection of type (b).

We use GUROBI[4] as ROCKIT's internal ILP solver. According to Koch et al. [KAA+11], who executed different ILP systems on standard ILP benchmarks, GUROBI is currently the fastest available ILP solver. Furthermore, it parallelizes the branch and bound algorithm and uses symmetry detection heuristics. For the latter one, they implemented orbit branching [OLRS11, OLRS07] with further non-published optimizations. We refer to Section 3.4.1 for a brief overview of symmetry detection approaches. Furthermore, they offer a free academic license and a frequent mailing list support. In ROCKIT we only initiate one ILP for each MAP state computation. This is then enriched with additional violated constraints in every cutting plane inference iteration. This requires much less execution time than initiating a new ILP instance for every CPI iteration.

A typical input parameter of current ILP solvers like GUROBI is the integrality gap parameter specifying an approximate solution's worst-case precision. This parameter provides the ILP solver with the maximum ratio between the objective value of the integer program and its relaxation. Thus, it guarantees a particular solution quality. Users of ROCKIT can set this `gap` parameter to values ranging from 10^{-1} to 10^{-10}.

6.2 Benchmarks

We use the following established benchmark MLNs for empirical evaluation. As these MLNs have been used in several previous publications, we ensure comparability to existing results.

[4] `http://www.gurobi.com/`

Entity Resolution (ER) Entity resolution is the problem of finding records which belong to the same real-world entities. Singla and Domingos [SD06a] combined several existing approaches for entity resolution. The benchmark they created for their experiments consists of 10 predicates where 4 of them are hidden predicates. In total there exist 7,720 formulas where 1,276 have a weight which is not zero. It has 12,892 evidence atoms and the completely grounded Markov logic network consists of 390,720 clauses.

Information Extraction (IE) In information extraction the task is to extract database records from text or semi-structured sources. The benchmark is taken from the experimental section of [PD07]. In their work they performed the segmentation of all records and entity resolution together in a single integrated inference process. The benchmark has 18 predicates where 2 of them are hidden. It has 1,024 formulas each of them having a weight not equal to zero. Furthermore, we obtain 258,079 evidence axioms and 340,737 clauses after a complete grounding.

Link Prediction (LP) This benchmark has been the first Markov logic benchmark which was created. It is used in the experimental section of the famous Markov logic publication [RD06]. It models the connections of different activities from a university department with its faculty, staff, and students [RD06]. The task is then to find who is advised by whom. This benchmark has only 1 hidden and 21 observed predicates and 24 formulas. Although the number of formulas is relatively small compared to other benchmarks, the complexity of the formulas is comparable high. The LP benchmark contains 1,031 evidence axioms and evolve to 354,587 clauses if grounded.

Protein Interaction (PR) The protein interaction (PR) benchmark describes interactions with proteins, enzymes, and phenotypes. To the best of our knowledge the benchmark was created within the genetic interaction extraction challenge [Néd05]. One of six teams applied Markov logic and achieved the best result in the competition. The benchmark has the highest number of formulas (2,503 formulas, thereof 2,461 formulas with non-zero weight) and the highest number of ground clauses (40,234,321) of all five benchmarks. Furthermore, it consists of 5 observed and 2 hidden predicates and 1,031 evidence atoms.

Relational Classification (RC) The relational classification benchmark performs classification on the CORA [MNRS00] dataset. In [NRDS11] the MLN was used to compare the performance of TUFFY to the ALCHEMY system. With 4 predicates (thereof 1 hidden predicate), 17 formulas, 99,161 evidence axioms, and 202,215 ground clauses the benchmark is comparably small. However, the formulas have a relatively high complexity.

	ER	IE	LP	PR	RC
predicates	10	18	22	7	4
thereof hidden	4	2	1	2	1
formulas	7,720	1,024	24	2,503	17
thereof $w \neq 0$	1,276	1,024	24	2,461	17
evidence	12,892	258,079	1,031	12,999	99,161
clauses	390,720	340,737	354,587	40,234,321	202,215

Table 6.1: Some characteristics of the ML benchmark datasets [NNS13].

The ER, IE, LP, and RC datasets were taken from the TUFFY web-page. These datasets were used in the publication [NRDS11]. The PR dataset was directly downloaded from the ALCHEMY website and weights were learned with ALCHEMY. Table 6.1 summarizes the properties of the five benchmarks.

6.3 Experimental Setup

In our experiments we compare different system configurations of ROCKIT and other MLN systems with respect to their runtime and their quality of the resulting approximate MAP state (interpretation). For measuring the quality of an interpretation we compute its objective. The objective of an interpretation x is the non-normalized sum of the number of true groundings $n_i(\mathbf{x})$ of all clauses F_i in $\mathbf{x}$.

$$Obj(x) = \sum_i w_i n_i(x)$$

We refer to Section 2.3 for further details and a formal definition of Markov logic. Intuitively speaking, the interpretation x has a higher quality than another interpretation y, if its objective $Obj(x)$ is higher than $Obj(y)$.

For our experiments we implemented a separated component, which computes the objective of an interpretation without constructing an ILP problem. This component first reads a systems approximate MAP state and generates database tables as described in Section 4.2. Then, it builds one SQL query per formula which counts all non-violated groundings and multiplies the result with the respective weight of the first-order formula. The sum over all formulas, finally, results in the objective.

Out of the objective, we compute the relative error ϵ of a system's solution. The relative error is a common way of analyzing the quality of approximate approaches [KF09]. We say that a specific MAP state x reaches a certain relative error ϵ if

$$Obj(x) \geq Obj(x^*) \cdot (1 - \epsilon)$$

where x^* is the optimal MAP state and thus $Obj(x^*)$ is the highest possible objective.

In our experiments, we measured the time needed to compute an interpretation whose objective has a relative error of *at most* $\epsilon = 10^{-\tau}, \tau \in \{1, 2, 3, 10\}$, with respect to the optimal objective. For determining this optimal objective we used ROCKIT to compute an ILP solution x_{10} whose objective value has a relative error of at most 10^{-10} and computed the actual objective $Obj(x_{10})$ of the interpretation corresponding to this ILP solution. From this value we computed $Obj(x_\tau)$ for $1 \leq \tau \leq 3$ by multiplying $Obj(x_{10})$ with $1 - 10^{-\tau}$. For finding the solution x_{10} we set the relative `gap` parameter of ROCKIT to 10^{-10}.

An exception was the LP benchmark, where no system could generate a solution with a relative error ϵ of 10^{-10} or 10^{-3}. To that end, we computed $Obj(x_2)$ for $\epsilon = 10^{-2}$ with ROCKIT and determined $Obj(x_1)$ by multiplying $Obj(x_2)$ with $(1 - 10^{-1})/(1 - 10^{-2})$.

The MLN systems were run, for each $\tau \in \{1, 2, 3, 10\}$, with an increasing number of MaxWalkSAT flips for ALCHEMY and TUFFY or with decreasing values of Gurobi's relative `gap` parameter for MARKOV THEBEAST and ROCKIT until a parameter configuration achieved an interpretation weight of at least ω_τ, or until one hour had passed, whichever came first.

All experiments were performed on a virtual machine with 8 GB RAM and 2 cores with 2,4 Ghz, unless otherwise stated.

6.4 Experimental Results

In the following experimental results, we empirically show that our CPA method (a) reduces the number of ILP constraints, (b) decreases runtime, and that (c) ROCKIT outperforms the state-of-the-art ML systems ALCHEMY, MARKOV THEBEAST, and TUFFY (version 0.3) [NRDS11][5] both in terms of efficiency and quality of the results. Furthermore, we will show that (d) the runtime of ROCKIT with the CPA method declines with the number of cores.

6.4.1 The CPA Method Reduces the Number of ILP Constraints

Let us first turn our attention to the question whether our CPA algorithm reduces the number of ILP constraints irrespective if it is combined with the CPI approach or not. We address that question by counting the ILP constraints for each benchmark and for four different configurations: without CPI and without CPA, without CPI and with CPA, with CPI and without CPA, as well as with CPI and with CPA.

Furthermore, we experimentally show that our CPA algorithm does not only aggregate simple clauses containing exactly one literal, but is also aggregating complex clauses consisting of more than one literal. To that end, we categorize the number of counting constraints in simple and complex counting constraints whenever we applied our CPA approach. Since one aggregated constraint might

[5] http://research.cs.wisc.edu/hazy/tuffy/

aggregate either just a few or very many constraints, we also compute the percentage of constraints which has been aggregated in simple or complex counting constraints.

Table 6.2 lists the number of ILP constraints with/without CPI and with/without CPA and provides details about simple and complex counting constraints. When we apply neither CPI nor CPA the number of ILP constraints equals the number of (ground) clauses in Table 6.1 since our translation requires exactly one constraint per ground clause.

The number of ground clauses always decreases significantly when we apply the CPA algorithm. Let us first examine the decrease of constraints with CPA in the case where we disable CPI. There, the highest absolute decrease is measured for the PR benchmark where the number of constraints is reduced from $40,234,321$ to $1,404,026$ constraints. The highest relative difference exists in the LP benchmark, where we get $9,401$ constraints with CPA and $354,587$ without applying CPA which is 38 times larger. The benchmark with the lowest relative difference is the IE benchmark. It has 4.8 times fewer constraints if CPA is applied. Let us now turn our attention to the case where we combine CPI with CPA. Here, the highest absolute and the highest relative decrease are both measured on the PR dataset where the constraints decreased from $2,688,122$ to $1,573$. Thus, with CPA we had 1709 times fewer number of constraints then without CPA. The IE benchmark has the lowest relative decrease of constraints which is $4,041/932 = 4.3$ times lower with CPA.

As a side effect, we also experimentally prove that the number of constraints always decreases if we apply cutting plane inference. This observation is made for the case where CPA is disabled and enabled. In case CPA is disabled, the highest absolute decrease was measured for the PR dataset where we obtained $40,234,321$ constraints without CPI and $2,688,122$ with CPI. In case CPA is enabled, we again measured the highest absolute decrease for the PR dataset, where we get $1,404,026$ constraints without CPA and $1,573$ with CPA.

In summary, the lowest number of constraints were achieved when CPI and CPA are combined. We obtain at least $202,215/10,064 = 22$ times (RC benchmark) at at most $40,234,321/1,573 = 25,578$ times (PR benchmark) lower number of constraints with CPI and with CPA compared to disabling CPI and disabling CPA.

We now examine the complexity of the constraints which are aggregated with CPA. Let us first define some upper numbers for counting constraints with only one literal which we called simple counting constraints. In general, the number of simple counting constraints can be as most as large as the number of formulas with a weight not equal zero because we can aggregate all ground clauses with only one single literal to one counting constraint and we compute the aggregation separate for each first order formula. The reader is encouraged to compare the number of formulas in Table 6.1 with the number of simple counting constraints in Table 6.2.

In almost all datasets we aggregated some complex ground clauses. The only exception is the PR benchmark where we result in no complex counting constraints

	ER	IE	LP	PR	RC
w/o CPI w/o CPA	390,720	340,737	354,587	40,234,321	202,215
w/o CPI w/ CPA	24,986	70,944	9,401	1,404,026	21,074
thereof $\diamond$ counting constraints					
- $\diamond$ = simple	1,250	1,013	15	2,499	14
	(85%)	*(38%)*	*(12%)*	*(21%)*	*(39%)*
- $\diamond$ = complex	23,736	69,931	9,386	1,401,527	21,060
	(15 %)	*(62%)*	*(88%)*	*(79%)*	*(61%)*
w/ CPI w/o CPA	357,056	4,041	31,658	2,688,122	164,047
w/ CPI w/ CPA	10,782	932	6,617	1,573	10,064
thereof $\diamond$ counting constraints					
$\diamond$ = simple	719	481	17	1,573	9,772
	(89%)	*(94%)*	*(14%)*	*(100%)*	*(53%)*
$\diamond$ = complex	10,063	451	6,600	0	22
	(11%)	*(6%)*	*(86%)*	*(0%)*	*(47%)*

Table 6.2: Number of ILP constraints generated by ROCKIT with and without CPI as well as with and without CPA (inspired by [NNS13]). Simple counting constraints aggregate clauses with exactly one literal whereas complex counting constraints aggregate clauses consisting of at least two literals. The per cent number below indicates the relative amount of clauses which were aggregated with simple or complex counting constraints, respectively. For determining these numbers we applied the relative error 10^{-10} for ER, IE, RC, and PR benchmarks and the relative error 0.01 for the LP benchmark.

with CPI and with CPA. Without CPI, we have a relatively high number of complex counting constraints of $1,401,527$. 79% of all constraints have been aggregated in one of the $1,401,527$ counting constraints while 21% were aggregated in one of the $2,499$ simple counting constraints. Obviously, none of the complex ground clauses has been violated within the cutting plane inference loops. This property of the PR benchmark also leads to the higher relative reduction of ILP constraints compared to other benchmarks.

6.4.2 CPA Decreases Runtime

After showing that our CPA method reduced the number of ILP constraints significantly we now inspect how this reduction affects runtime. To this end, we again compare the runtime for the four different ROCKIT configurations. For each configuration and each benchmark we measured the time needed to compute an interpretation whose objective has a relative error of *at most* $\epsilon = 10^{-\tau}, \tau \in \{1, 2, 3, 10\}$, with respect to the optimal objective. We refer the reader to Section 6.3 for a detailed description how we determined the relative error of each benchmark.

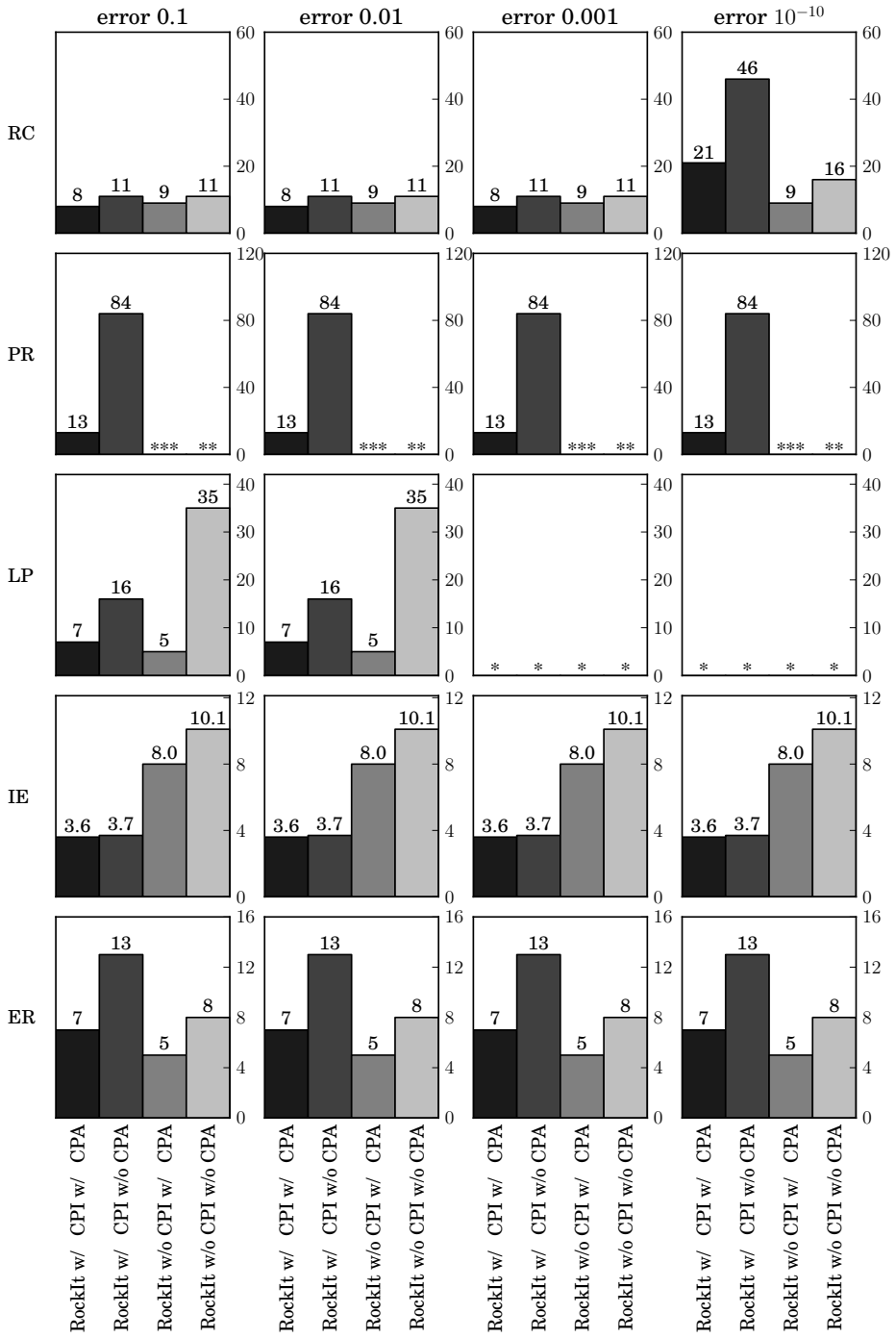

Figure 6.2: Comparison of runtimes per second of ROCKIT with/without CPI and with/without CPA (* gap not reached in 1 hour, ** out of memory, *** did not terminate within 1 hour). Applying CPA is always faster than not applying CPA.

Figure 6.2 displays the runtime results in seconds for every benchmark, every relative error, and every configuration. When we compare the runtimes of the configuration 'with CPI and with CPA' with the runtimes 'with CPI and without CPA' we notice that runtimes are always lower when we enabled CPA except for the IE benchmark where runtimes are almost equivalent. The largest difference is measured for the PR benchmark where the runtime without CPA is 84 seconds and with CPA is 13 seconds. The same applies when we compare the two configurations where CPI is disabled. Runtimes are always higher 'without CPI and with CPA' compared to the configuration 'without CPI and without CPA'. For the RC benchmark we measure the lowest difference of 3 seconds for the relative errors $\epsilon = \{10^{-1}, 10^{-2}, 10^{-3}\}$. The largest difference occurs in the LP benchmark where we obtain a runtime of 35 seconds without CPA and a runtime of 5 seconds with CPA. This huge runtime decrease is due to much lower solving times of the ILP. The runtime of the SQL queries have been marginal.

In our experiments we observed a tendency that the difference of the runtime between enabling and disabling our CPA approach was higher, when more constraints were aggregated. The reader is encouraged to compare the runtimes of Figure 6.2 with the number of constraints in Table 6.2. However, one would have to examine the effect of CPA on many more datasets to speak of a clear correlation.

As a side effect our experiments also examined the effect of disabling or enabling CPI. For the PR benchmark we were not able to compute any solution without CPI since the MLN became intractable large. In particular, we got an out of memory exception if we also disabled CPA and were not able to solve the MLN within one hour if we enabled CPA. With CPI runtimes were faster for the IE benchmark and for the LP benchmark (both without CPA). Almost equal runtimes were achieved for the RC benchmark and for the LP benchmark (both with CPA). Lower runtimes were measured for the ER benchmark. The explanation of these lower runtimes with CPI are that almost all constraints are violated and thus we have to add almost as many constraints with CPI than without CPI. This results in lower runtimes since CPI needs to query for violated constraints at least twice and might have to perform more than one CPI iteration. Overall, we still strongly recommend using CPI because if CPI is slower the loss of runtime is usually low. However, for some problems CPI can compute a MAP-state which would be intractable without CPI.

6.4.3 ROCKIT Outperforms Other Markov Logic Systems

In this section we present the results of the comparison of our system ROCKIT with other Markov logic systems MARKOV THEBEAST, TUFFY, and ALCHEMY. These systems are presented in Section 5.3. For our system ROCKIT we used the configuration with CPI and with CPA. We refer to Section 6.4.2 for details on alternative configurations and their runtime results. As described in the experimental setup in Section 6.3, we measured the time needed to compute an interpretation whose objective has a relative error of *at most* $\epsilon = 10^{-\tau}, \tau \in \{1, 2, 3, 10\}$, with respect to

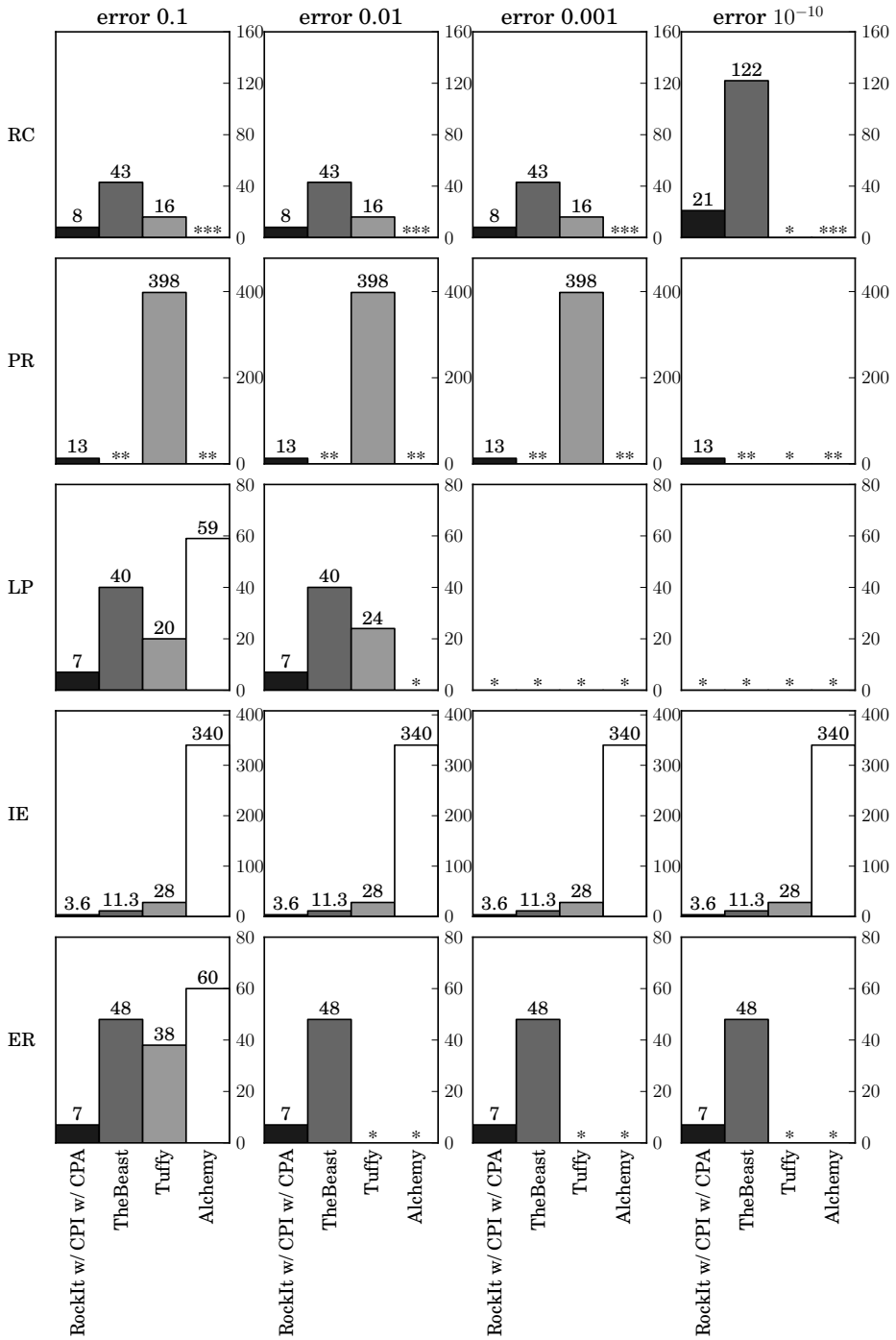

Figure 6.3: Comparison of runtimes per second of ROCKIT and state-of-the art MLN engines TUFFY, MARKOV THEBEAST, and ALCHEMY for different gaps (* gap not reached in 1 hour, ** out of memory, *** did not terminate within 1 hour). ROCKIT is always faster and reaches a higher error for some datasets. (inspired by [NNS13])

the optimal objective for each MLN system and each benchmark.

The results for every benchmark, every relative error, and every system are visualized in Figure 6.3. ROCKIT is more efficient and is always able to compute a higher objective than TUFFY, MARKOV THEBEAST, and ALCHEMY.

For the ER benchmark only the systems MARKOV THEBEAST and ROCKIT can generate solutions with a relative error lower than $\epsilon = 10^{-1}$. ROCKIT needs 7 seconds to compute a MAP state reaching all relative errors while MARKOV THEBEAST needs 48 seconds. TUFFY and ALCHEMY can only generate a solution satisfying the error $\epsilon = 10^{-1}$ within 38 seconds and 60 seconds, respectively. In the IE benchmark, every system reach the lowest error $\epsilon = 10^{-10}$. ROCKIT requires 3.6 seconds, MARKOV THEBEAST requires 11.3 seconds, TUFFY requires 28 seconds, and ALCHEMY requires 340 seconds. On the LP benchmark no system is able to generate a MAP state with relative error $\epsilon = 10^{-3}$ or lower. ALCHEMY is also not able to reach $\epsilon = 10^{-2}$ on this dataset. The runtimes of the systems ROCKIT, MARKOV THEBEAST, TUFFY, and ALCHEMY are 7 seconds, 40 seconds, 20 seconds, and 59 seconds, respectively. The highest performance difference between ROCKIT and the other MLN systems is measured for the PR dataset. While MARKOV THEBEAST and ALCHEMY can not generate any solution because they ran out of memory, TUFFY needs 398 seconds for computing a solution. Contrary to this, ROCKIT only needs 13 seconds to compute a solution which error was lower or equal than 10^{-10}. Last, we turn our attention to the RC dataset. ALCHEMY could not find any solution within one hour. TUFFY did not reach the relative error $\epsilon = 10^{-10}$. For computing a solution with a relative error of $\epsilon = 10^{-3}$ TUFFY needs 16 seconds. The fastest runtimes for ROCKIT and MARKOV THEBEAST for generating a MAP state with an error lower than $\epsilon = 10^{-10}$ are 8 seconds and 43 seconds, respectively.

In all cases, ROCKIT was able to compute a higher weighted solution in less time. Overall, TUFFY showed the second best performance, directly followed by MARKOV THEBEAST in our experiments. The highest runtimes were measured for ALCHEMY.

We conjecture that ROCKIT with CPI and without CPA is more efficient than MARKOV THEBEAST because of ROCKIT's more compact ILP formulation and the use of MySQL as database system. The runtimes of ROCKIT with CPI and without CPA are visualized in Figure 6.3.

6.4.4 Runtime Declines With the Number of Cores

The last issue we address in these experiments is the question weather the parallelization of the MAP query leads to runtime improvements. To that end, we examine if runtime declines with the number of cores. In particular, we run ROCKIT with CPI and with CPA on the same virtual machine with 8 GB RAM with a varying number of 2,4 Ghz cores.

Figure 6.4 summarizes the runtime comparison for 1, 2, 4, and 8 cores. For each benchmark, the runtime decreases when the number of cores increases with

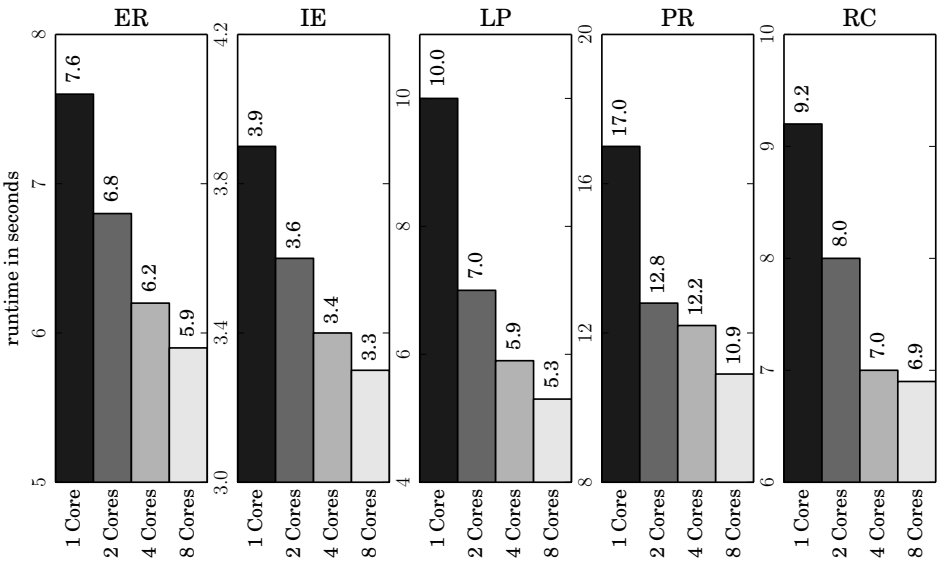

Figure 6.4: Performance improvements of ROCKIT with CPA on multiple cores [NNS13]. We show the results for the lowest gap 10^{-10} for ER, IE, RC, and PR benchmarks and the gap 0.01 for the LP benchmark.

a diminishing reduction in runtime. Tendentielly, the largest runtime decreases are achieved between the runtimes between 1 core and 2 cores, the second largest runtime decreases are between 2 and 4 cores and the least runtime decreases are measured between 4 and 8 cores. A counter example is the PR benchmark, where the runtime between 2 cores and 4 cores decreases 0.6 seconds while we measured a large runtime decrease of 1.3 seconds between 4 and 8 cores.

When comparing the runtimes of 1 and 8 cores, the ER benchmark has a relative runtime decrease of 22%, the IE benchmark has a runtime decrease of 15%, the LP benchmark has the highest relative decrease of about 53%, the PR benchmark's relative decrease is 36%, and the RC benchmark has a decrease of 25%. The high runtime decrease for the LP benchmark is because the aggregation of clauses with more than one literal can be efficiently processed in parallel.

Chapter 7

Conclusion and Future Work

Section 7.2 draws a conclusion over the whole Part I of this thesis in which we address the research questions mentioned in Section 1.2 and Section 7.3 points out future work directions. Additionally, Section 7.1 provides a comprehensive example which covers the most important concepts of CPA in Part I.

7.1 Concluding Example

In order to quickly being able to gasp the main contribution of Part I, we provide an encompassing example. The example intuitively illustrates the basic idea behind our novel cutting plane aggregation method (see Section 3.4) and its difference to our novel optimized ILP translation (see Section 3.2). The reader should be familiar with Markov logic as introduced in Section 2.3 and integer linear programming explained in 2.4 to understand this example.

Previous examples were kept simple in a way that they aggregated only clauses with one literal. In this section we construct a example which shows how our CPA method (a) aggregates clauses with more than one literal, (b) exploit evidence, and (c) can aggregate transitivity formulas if evidence occurs. Transitivity formulas present a challenge for symmetry detection approaches [Rie08].

However, please note that for brevity reasons important issues like negative weights, the concepts of cutting plane inference and its integration with our CPA algorithm (Section 3.3 and Section 3.5.1), and leveraging relational database systems for grounding (Chapter 4) are not covered in this example.

We start with defining a MLN with one transitive formula, the ground clauses, and the given evidence.

Example 18. *We define a MLN with only one formula which expresses the transitivity of the friends relationship.*

$$1.1 \quad friends(x, y) \wedge friends(y, z) \Rightarrow friends(x, z)$$

When we transform this formula to the conjunctive normal form, we get:

$$1.1 \quad \neg friends(x, y) \vee \neg friends(y, z) \vee friends(x, z)$$

Furthermore, let us assume that we retrieved the following n ground clauses:

$$\begin{array}{lccccc} 1.1 & \neg friends(A,B_1) & \vee & \neg friends(B_1,C) & \vee & friends(A,C) \\ 1.1 & \neg friends(A,B_2) & \vee & \neg friends(B_2,C) & \vee & friends(A,C) \\ & & & \dots & & \\ 1.1 & \neg friends(A,B_n) & \vee & \neg friends(B_n,C) & \vee & friends(A,C) \end{array}$$

If we have evidence that A and all B_is are friends

$$friends(A,B_i) \quad i \in \{1,\dots,n\}$$

our retrieved ground clauses reduce to:

$$\begin{array}{lccc} 1.1 & \neg friends(B_1,C) & \vee & friends(A,C) \\ 1.1 & \neg friends(B_2,C) & \vee & friends(A,C) \\ & \dots & & \\ 1.1 & \neg friends(B_n,C) & \vee & friends(A,C) \end{array}$$

Please note, that the retrieved ground clauses in Example 18 can either result from all possible groundings or originate from violated constraints within a cutting plane inference iteration. How we ground a MLN if evidence is considered is explained in Section 2.3.2. Now let us transform the ground clauses into ILP constraints as it is described in Section 3.1.

Example 19. *Let us assume that the MLN and the retrieved ground clauses from Example 18 is given. We introduce a binary ILP variable for each atom and one binary z_i variable per ground clause. For presentation reasons, we use the same names for literals than for ILP variables. When we now apply the standard ILP translation of Section 3.1, we result in the following ILP.*

$$\begin{array}{c} \max 1.1z_1 + \dots + 1.1z_n \\ (1 - friends(B_1,C)) + friends(A,C) \leq z_1 \\ (1 - friends(B_2,C)) + friends(A,C) \leq z_2 \\ \dots \\ (1 - friends(B_n,C)) + friends(A,C) \leq z_n \end{array}$$

We result in n ILP constraints and n summands in the objective.

In the standard ILP translation as illustrated in Example 19, we define exactly one ILP constraint and one summand in the objective for each ground clause. Since the weight of our ground clause is positive, the z variable will always *try* to become one. Thus, the ILP constraint must force the z variable to become zero, if the ground clause evaluates to *false*. If the ground clause has a negative weight, we need to introduce an ILP constraint which forces the z variable to become one, if the ground clause evaluates to *true*, respectively. For details, we refer the reader to Section 3.1. This translation is novel since it requires only one ILP constraint per ground clause, irrespectively if the weight is positive or negative.

If we now apply our CPA strategy described in Chapter 3, we can significantly reduce the number of the summands in the objective and required ILP constraints.

Example 20. *We again use the MLN and the retrieved ground clauses from Example 18. We also introduce a binary ILP variable for each atom and use the same variable name. This time, we introduce only one integer variable z. If we now apply our novel CPA method, we result in the ILP:*

$$\max 1.1z$$
$$z \leq n$$
$$n \cdot friends(A,C) + (1 - friends(B_1,C)) + \ldots + (1 - friends(B_n,C)) \leq z$$

We now require only 2 *ILP constraints and* 1 *summand in the objective.*

Due to the CPA method, we could reduce the required ILP constraints from n (Example 19) to 2 (Example 20) and the summands in the objective from n to 1, respectively. This was possible, because we could aggregate every retrieved ground clause to one so-called counting feature. This counting feature is then represented by one integer variable z which is in this example the only required summand in the objective. Furthermore, the counting feature translates to two ILP constraints. The first constraint ensures that the z variable is not larger than the number of aggregated clauses. In combination with the second constraint, the z variable becomes n if $friends(A,C)$ is *true*. If $friends(A,C)$ is *false*, the z variable *counts* the number of *true* $friends(A,B_i)$ atoms. We refer the reader to Chapter 3 for details.

Please note, that we omitted the trivial translation of the evidence axioms into the ILP for clarity reasons. In order to encode evidence, we have to add additional n ILP constraints of the form

$$friends(A,B_i) \geq 1 \quad \forall i \in \{1,\cdots,n\}$$

in Example 19 and Example 20.

7.2 Conclusion

In the first chapter of this thesis we presented the novel cutting plane aggregation (CPA) algorithm which improves the translation of maximum a-posteriori (MAP) queries to integer linear programs (ILPs). For formal definitions of MAP queries and ILPs we refer to Section 2.1.3 and Section 2.4, respectively. On the one hand the CPA algorithm leads to fewer ILP constraints and fewer ILP variables and on the other hand it makes symmetries explicit to state-of-the-art ILP solvers.

Within this conclusion we recall and answer the research questions which we set up in Section 1.2.1:

Q1 *Can we improve existing ILP translation techniques such that we reduce the size of the ILP and make the symmetries of the model more explicit to symmetry detection heuristics?*

Existing work requires more than one constraint for translating one ground clause to an integer linear program. In the translation of [Rie08, Wil99], one

weighted ground clause consisting of e.g. three literals results in four linear constraints. Riedel [Rie08] suggested cutting plane inference (CPI) for solving MAP queries. In CPI we solve several smaller ILPs which often leads to significant runtime reductions. Hereby, we add only those constraints to the ILP which are violated by the current solution and solve the ILP. This is repeated until no violated constraints remain. The resulting MAP state is equivalent to the MAP state received without CPI. We refer to Section 3.3 for details.

In Chapter 3 we introduced two improved translation techniques. First, we showed in Section 3.2 that it is possible to translate every ground clause to exactly one ILP constraint irrespectively if it is weighted or unweighted.

Second, we provide an aggregation technique called cutting plane aggregation in Section 3.4 which allows to aggregate more than one ground clause to one ILP constraint. Within this translation, we exploit symmetries on the ground clause level and aggregate clauses using counting constraints. This makes symmetries explicit to the symmetry detection mechanisms of state-of-the-art ILP solvers. We refer to Section 3.4.1 for details. Since the CPA method operates on ground clauses we may call this a *bottom-up approach*. Contrary to this, most approaches in the related field of lifted inference (see Section 5.1) discover symmetries on the first-order level which can be seen as *top-down approach*. Our approach is the first aggregation approach which tightly integrates the cutting plane inference (CPI) methodology with the CPA approach. For details about the CPI method please read Section 3.3. With respect to our aggregation technique, we only *estimate* the optimal aggregation scheme avoiding a costly exact computation in each CPI iteration. The algorithm used for this estimation is described in Section 3.4.2. We experimentally discovered that exact approaches neutralize the runtime benefits gained by the CPA concept.

Q2 *How can we parallelize the solution process and tightly integrate relational database management systems (RDBMS) within this process?*

We can parallelize important parts of the MAP query solving process, since our aggregation technique can be processed for each first-order logic formula separately. In particular, we parallelize the computation of the violated constraints within each cutting plane inference iteration and the algorithm for estimating the optimal aggregation scheme. Furthermore, we use the capability of state-of-the-art ILP solvers like GUROBI to solve ILPs in parallel. The complete parallelization pipeline is depicted in Figure 3.1 and described in Section 3.5.2. For a very brief sketch of the properties of the solver GUROBI we refer to Section 6.1.

Riedel [Rie08] was the first who showed that relational database management systems can be efficiently used for solving MAP queries. However, he only briefly sketched this technique. In this thesis, we exactly define how the evidence of MLNs are stored in database tables and we provide exact algorithms how efficient SQL queries are constructed. Chapter 4 contains SQL query creation algorithms for determining the ground clauses of a MLN (Section 4.1), finding the violated constraints within a CPI iteration (Section 4.2), and computing the counting features

for CPA (Section 4.3). For the latter one we finally preferred an implementation within Java because of better possibilities of evidence consideration. The interested reader is referred to Section 4.3 for details.

Q3 *Does our new techniques reduce runtime and outperform existing Markov logic systems with respect to runtime and quality of the results?*

The research question Q3 is answered empirically in Section 6.4. For the experiments we used five established Markov logic benchmarks. Details about the origin of these benchmarks and its properties are described in Section 6.2. Our experiments show that the number of ILP constraints is often drastically reduced when applying the CPA algorithm. We also showed, that the CPA approach aggregated a significant number of non-trivial constraints consisting of more than one literal. Runtimes were always faster with CPA than without CPA except in one case, where runtimes were almost equal. Furthermore, there is a weak tendency that the relative runtime decrease is higher if the relative ILP constraint reduction is higher. As a side effect, we also show a reduction of constraints and runtime improvements in most cases with CPI.

Furthermore, we compared our approach against the three state-of-the-art Markov logic solvers MARKOV THEBEAST, ALCHEMY, and TUFFY with respect to runtime and quality of the solution. As quality measure we use the relative error which is described in Section 6.3. The quality of the solution is higher, if the measured relative error of the solution is lower. On all benchmarks our Markov logic solver ROCKIT reached a lower relative error in less time than the solvers MARKOV THEBEAST, ALCHEMY, and TUFFY.

7.3 Future Work

There are several further directions for future work. In the following, we select a few promising extensions.

Extension of the aggregation strategy Currently, our cutting plane aggregation algorithm allows that only one literal is different while all other literals must be equal. It is possible to extend cutting plane aggregation such that clauses with two different literals can be aggregated. However, the resulting optimization problem would be quadratic. Thus, we either would have to apply quadratic solving strategies or we would have to *linearisize* the problem by applying *and* linearization [RRR06].

Integration of lifted inference algorithms In the last years more and more work in lifted inference arose. While most work focus on lifting marginal inference, some of this work also lift maximum a-posteriori inference. Future investigations include the questions if such approaches can be combined with our

CPA approach. Promising starting points for possible integrations are the approaches [AB12, BHR13]

Generalization of the CPA approach Currently, we presented CPA for Markov logic. However, the basic idea of cutting plane aggregation can be generalized to detect symmetry groups in arbitrary integer linear programs. For significant effect of CPA, the ILPs have to follow certain characteristics like often have the same objective weights for many variables. To that end, a first step is to investigate the aggregation potential of real-world ILP problems.

Extension of our ROCKIT solver For MAP inference we experimentally verified that our ROCKIT solver is currently the fastest existing solver. However, we only implemented basic algorithms for marginal inference and learning which have not yet been evaluated against other systems. For improving marginal inference, the integration of existing symmetry detection algorithms [Nie12, Nie13] into ROCKIT and their further development are possible next steps. For parameter learning, a starting point for further implementations is the work from Lowd and Domingos [LD07].

Part II

Application in Description Logics

As far as the laws of mathematics refer to reality, they are not certain;
and as far as they are certain, they do not refer to reality.
(Albert Einstein)

Chapter 8

Log-Linear Description Logics

In real-world applications, uncertainty occurs often in form of *degrees of confidence* or *trust*. The semantic web community, for instance, has developed numerous data mining algorithms to generate confidence values for description logic axioms with ontology learning and matching being two prime applications [EMS+11, Bre06, WLB12]. Most of these confidence values have no clearly defined semantics. Confidence values based on lexical similarity measures, for instance, are in widespread use while more sophisticated algorithms that generate actual probabilities make often naïve assumptions about the dependencies of the underlying probability distribution. Hence, formalisms and inference procedures are needed that incorporate degrees of confidence in order to represent uncertain axioms and to compute answers to probabilistic queries while utilizing the logical concepts of *coherency* and *consistency*. [NNS11]

To respond to this need, we introduce log-linear description logics as a novel formalism for combining deterministic and uncertain knowledge. We describe a convenient representation of log-linear description logics that allows us to adapt existing concepts and algorithms from statistical relational AI to answer standard probabilistic queries. In particular, we formulate maximum a-posteriori queries and present an efficient algorithm that computes most probable *coherent* models, a reasoning service not supported by previous probabilistic description logics. [NNS11]

The presented reasoning approach for log-linear DLs is restricted to DLs where consequence-driven reasoning is possible. An incomplete list of such DLs can be found in Section 8.1. Log-linear description logics are thus a combination of description logics and log-linear models. We refer the reader to Section 2.1.1 for an introduction of log-linear models and to Section 2.5 for details about description logics. The syntax of log-linear description logics is that of the underlying description logic where it is possible (but not necessary) to assign real-valued weights to axioms. The semantics is defined by a log-linear probability distribution over *coherent* ontologies. The syntax and semantic are introduced in Section 8.2. The fact that we only consider coherent worlds is one of the differences to Part I of

this thesis. In principle, log-linear description logics can be mapped to a *concrete* Markov logic network, which models the coherency with hard rules and allows to assign weights to each description logics axiom. Thus, Part II is an application of Part I. Section 8.3 guides through the steps which are necessary to transform a description logic to its corresponding log-linear description logic and, finally, to a Markov logic network. In this context, we also define the novel query which asks for the most probable *coherent* ontology.

As we will learn, we require the ability to weight conjunctions of literals in order to correctly translate the log-linear description logic semantic to a Markov logic networks. However, Markov logic only supports weighted clauses. If the conjunctive normal form of a first-order logic formula consists of more than one clause, Markov logic solvers usually apply heuristics, which split them into multiple weighted clauses. However, this leads to a different semantic. Thus, Section 8.4 presents an extension of Markov logic such that it can deal with weighted conjunctions. In fact, we even further generalize the Markov logic syntax so that it works with weighted formulas in conjunctive normal form. Furthermore, we extend our cutting plane aggregation approach of Part I such that it can aggregate conjunctions as well.

Generally, this chapter answers the research questions Q4 and Q5 from Section 1.2.2:

Q4 *How can we combine log-linear models with description logics and define the query of a most-probable coherent ontology?*

Q5 *Can we efficiently compute the most-probable coherent ontology utilizing the theory of Part I?*

In particular, Section 8.2 and Section 8.3.1 until Section 8.3.4 address Q4 while Section 8.3.5 and Section 8.4 tackle Q5.

This chapter describes an extended version of our publications [NNS11] with several illustrative examples. We took text passages, pictures, and examples from our publications [NNS11, NN11]. However, we explain normalization in Section 8.3.1 much more exhaustive by using mainly the sources [BBL05a, BBL05b, BBL08]. Our representation in this thesis focuses on connecting Part I with Part II in Section 8.3.5 which is not contained in [NNS11, NN11]. Furthermore, Section 8.1 goes beyond the content of [NNS11] and is summarized from various different sources. Section 8.4 contains a not yet published idea of extending the ILP formulation to arbitrary conjunctions.

8.1 Possible Log-Linear DLs

Our novel log-linear description logic extends classical description logics. Each classical DL can be transformed to log-linear DL as long as consequence-driven

reasoning is supported [Krö10, BBL05a]. We refer the reader to Section 2.5.2 for details about consequence-driven reasoning.

In the following, we provide an incomplete list of such description logics. For details regarding naming conventions of description logics we refer the reader to [NB03].

- $\mathcal{ELRO}$ (also called $\mathcal{EL}^{++}$, OWL 2 profile EL) is able to perform scalable TBox reasoning. Classification can for example be performed in polynomial time. To that end, $\mathcal{EL}^{++}$ is used in many large medical ontologies [BBL05a, BBL08].

- $\mathcal{SROEL}(\sqcap, \times)$ is an extension of $\mathcal{EL}$ + + with local reflexivity (Self), conjunctions of roles, and concept products. Kroetsch [Krö10] proved that the corresponding calculus is in polynomial time.

- $\mathcal{SROIQ}$–RL (OWL 2 profile RL) is designed for applications which need a relatively high expressivity without resulting in too complex inference problems [BS13].

- $Horn$ – $\mathcal{SHIQ}$ is a restriction of the DL $\mathcal{SHIQ}$ which do not contain non-deterministic constructors such as positive disjunction. Compared to $\mathcal{EL}^{++}$, $Horn$ – $\mathcal{SHIQ}$ supports inverse roles and functional restrictions [Kaz09].

- $Horn$–$\mathcal{SROIQ}$ and $Horn$–$\mathcal{SHOIQ}$ are the Horn variants of the underlying description logics of OWL 1 DL and OWL 2 DL, respectively [ORS10]. For enabling consequence-driven reasoning they first translate the knowledge base in polynomial time into DATALOG^S which allows sets of constants and then again in polynomial time into DATALOG which heavily blows up the number of ground formulas.

- $\mathcal{ALCH}$ is the first DL which contains also non-Horn clauses for which a rule set have been found such that consequence driven reasoning is possible [SKH11].

For presenting the idea of log-linear description logics, we focus particularly on the description logic $\mathcal{EL}^{++}$ which captures the expressivity of numerous ontologies in the medical and biological sciences and other domains. The DL $\mathcal{EL}^{++}$ is also the underpinning of the web ontology language profile OWL 2 EL is based [BBL05a]. It is possible to express disjointness of complex concept descriptions as well as range and domain restrictions [BBL08]. In addition, role inclusion axioms (RIs) allow the expression of role hierarchies $r \sqsubseteq s$, reflexive roles $\epsilon \sqsubseteq s$, and transitive roles $r \circ r \sqsubseteq r$. We refer to Section 2.5.1 and Section 2.5.2 for the definition of the Syntax and Semantics of $\mathcal{EL}^{++}$ with range restrictions and without nominals and concrete domains.

Name	Syntax
top	$\top$
bottom	$\bot$
nominal	$\{a\}$
conjunction	$C \sqcap D$
existential restriction	$\exists r.C$
deterministic CBox $\mathcal{C}^{\mathsf{D}}$	
GCI$^{\mathsf{D}}$	$C \sqsubseteq D$
RI$^{\mathsf{D}}$	$r_1 \circ ... \circ r_k \sqsubseteq r$
RR$^{\mathsf{D}}$	$\mathsf{ran}(r) \sqsubseteq C$
uncertain CBox $\mathcal{C}^{\mathsf{U}}$	
GCI$^{\mathsf{U}}$	$\langle C \sqsubseteq D, w \rangle$
RI$^{\mathsf{U}}$	$\langle r_1 \circ ... \circ r_k \sqsubseteq r, w \rangle$
RR$^{\mathsf{U}}$	$\langle \mathsf{ran}(r) \sqsubseteq C, w \rangle$

Table 8.1: Syntax of the log-linear $\mathcal{EL}^{++}$ with range restrictions and without concrete domains.

8.2 Theory

Within this Section we explain the Syntax and the Semantics of log-linear description logics using the DL $\mathcal{EL}^{++}$. Please note again, that we can use every description logic for which consequence-driven reasoning is possible.

In Section 8.2.1, we describe how we assign weights to each description logic axiom in a way that some axioms are deterministic (they must hold) and some axioms are uncertain (they may hold). The semantic of log-linear description logics, described in Section 8.2.2, allows us to assign a probability to each possible set of axioms. This probability is zero if this set of axioms lead to an incoherent ontology or does not entail every deterministic axiom. Else, the probability depends on the sum of the weights of the entailed uncertain axioms. The higher this sum is, the higher is the probability of the given set of axioms.

8.2.1 Syntax

The syntax of log-linear description logics is equivalent to the syntax of the underlying description logic except that it is possible to assign weights to GCIs and RIs. More formally, a log-linear $\mathcal{EL}^{++}$ CBox $\mathcal{C} = (\mathcal{C}^{\mathsf{D}}, \mathcal{C}^{\mathsf{U}})$ is a pair consisting of a deterministic $\mathcal{EL}^{++}$ CBox $\mathcal{C}^{\mathsf{D}}$ and an uncertain CBox $\mathcal{C}^{\mathsf{U}} = \{\langle c, w_c \rangle\}$ which is a set of pairs $\langle c, w_c \rangle$ with each c being a $\mathcal{EL}^{++}$ axiom and w a real-valued weight assigned to c. In Table 8.1 we summarize the syntax of log-linear description logics. The only difference compared to Table 2.4 in Section 2.5 is the introduction of an additional uncertain CBox.

While the deterministic CBox contains axioms that are known to be true, the

uncertain CBox contains axioms for which we only have a *degree of confidence*. Intuitively, the greater the weight of an axiom the more likely it is true. This follows the same intuition as in Markov logic, since both inherit log-linear models. Every axiom can either be part of the deterministic or the uncertain CBox but not both. The deterministic CBox is assumed to be coherent.

As stated in Section 2.5 we can normalize each GCI and RI axioms. This is, of course, also possible for the uncertain axioms. Given a $\mathcal{EL}^{++}$ CBox $\mathcal{C}$ we use $\mathsf{BC}_{\mathcal{C}}$ to denote the set of basic concept descriptions occurring in $\mathcal{C}^{\mathsf{D}}$ or $\mathcal{C}^{\mathsf{U}}$. Details about normalization for log-linear description logics follows in Section 8.3.1.

Example 21 assigns weights to some of the axioms of ontology $\mathcal{O}_1$ which we defined in Example 7. If we interpret the assigned weights isolated, they mean that it is more likely that $Jaguar \sqsubseteq Brand$ holds than that $Jaguar \sqsubseteq Animal$ is true, which is again more likely than $Jaguar \sqsubseteq Cat$ is entailed in the solution.

Example 21. *We modify the Ontology $\mathcal{O}_1$ of Example 7 by assigning weights to some axioms. We call this log-linear ontology $\mathcal{O}_1^{LL}$:*

$Cat \sqsubseteq Animal$	*A cat is an animal.*
$Animal \sqcap Brand \sqsubseteq \bot$	*Something can not be a brand and a cat.*
$\langle Jaguar \sqsubseteq Cat, 0.5\rangle$	*A Jaguar is a cat.*
$\langle Jaguar \sqsubseteq Animal, 0.9\rangle$	*A Jaguar is an animal.*
$\langle Jaguar \sqsubseteq Brand, 1.2\rangle$	*A Jaguar is a brand.*

In this example, we defined the CBox $\mathcal{C} = (\mathcal{C}^{\mathsf{D}}, \mathcal{C}^{\mathsf{U}})$ with the deterministic CBox $\mathcal{C}^{\mathsf{D}} = \{Cat \sqsubseteq Animal, Animal \sqcap Brand \sqsubseteq \bot\}$ and the uncertain CBox $\mathcal{C}^{\mathsf{U}} = \{\langle Jaguar \sqsubseteq Cat, 0.5\rangle, \langle Jaguar \sqsubseteq Animal, 0.9\rangle, \langle Jaguar \sqsubseteq Brand, 1.2\rangle\}$. The set of basic concept descriptions is $\mathsf{BC}_{\mathcal{C}} = \{Animal, Brand, Cat, Jaguar\}$.

8.2.2 Semantics

The semantics of log-linear DLs is based on joint probability distributions over *coherent* $\mathcal{EL}^{++}$ CBoxes. The weights of the axioms determine the log-linear probability distribution. For a $\mathcal{EL}^{++}$ CBox $\mathcal{C} = (\mathcal{C}^{\mathsf{D}}, \mathcal{C}^{\mathsf{U}})$ and a CBox $\mathcal{C}'$ over the same set of basic concept descriptions and role names, we have that

$$P(\mathcal{C}') = \begin{cases} \frac{1}{Z} \exp\left(\sum_{\{\langle c, w_c\rangle \in \mathcal{C}^{\mathsf{U}} : \mathcal{C}' \vDash c\}} w_c\right) & \text{if } \mathcal{C}' \text{ is } \textit{coherent} \\ & \text{and } \mathcal{C}' \vDash \mathcal{C}^{\mathsf{D}}; \\ 0 & \text{otherwise} \end{cases}$$

where Z is the normalization constant of the log-linear probability distribution P[1].

The probability of a CBox $\mathcal{C}'$ is zero if $\mathcal{C}'$ is incoherent or if not all *hard* axioms from the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ are entailed in $\mathcal{C}'$. If $\mathcal{C}'$ is coherent and $\mathcal{C}' \vDash \mathcal{C}^{\mathsf{D}}$, then the probability is the normalized exponential sum over all weights w_c which

[1]Please note that we set $\sum_{\{\langle c, w_c\rangle \in \mathcal{C}^{\mathsf{U}} : \mathcal{C}' \vDash c\}} = 0$ if no c exist, and that we exclude this special case in our definition for clarification.

correspond to an axiom c. This is expressed by $\frac{1}{Z}\exp\left(\sum_{\diamond} w_c\right)$. All those axioms c must be entailed in $\mathcal{C}'$ and the axiom weight pair $\langle c, w_c\rangle$ must be an element in the uncertain TBox $\mathcal{C}^{\mathsf{U}}$. This is expressed with the sum condition $\diamond = \{\langle c, w_c\rangle \in \mathcal{C}^{\mathsf{U}} : \mathcal{C}' \models c\}$.

In this thesis, we are not interested in the exact value of the normalization constant Z since we aim at computing the most probable coherent world. For this computation, we need to find a coherent ontology with the highest probability, which is not influenced by dividing through a constant Z.

The semantics of the log-linear description logic $\mathcal{EL}^{++}$ leads to exactly the probability distributions one would expect under the open world semantics of description logics. Incoherent ontologies or those ontologies which do not entail all given hard axioms have the overall probability 0. All other ontologies have a probability greater than 0. We demonstrate these effects on the following example.

Example 22. *Let us recapitulate the weighted ontology $\mathcal{O}_1^{LL}$ from Example 21. We obtain a set of four basic concept descriptions $\mathsf{BC}_{\mathcal{C}} = \{Animal, Brand, Cat, Jaguar\}$. Our deterministic CBox is $\mathcal{C}^{\mathsf{D}} = \{Cat \sqsubseteq Animal, Cat \sqcap Brand \sqsubseteq \bot\}$ and the uncertain CBox is $\mathcal{C}^{\mathsf{U}} = \{\langle Jaguar \sqsubseteq Cat, 0.5\rangle, \langle Jaguar \sqsubseteq Animal, 0.9\rangle, \langle Jaguar \sqsubseteq Brand, 1.2\rangle\}$.*

We now compute the probability of different CBoxes C':

- *If $C' = \{Cat \sqsubseteq Animal\}$, we get the probability $P(C') = 0$ because $C' \not\models \mathcal{C}^{\mathsf{D}}$. This means that C' does not entail all axioms of the deterministic CBox $\mathcal{C}^{\mathsf{D}}$. In particular, the axiom $Cat \sqcap Brand \sqsubseteq \bot$ is not entailed.*

- *If $C' = \{Cat \sqsubseteq Animal, Cat \sqcap Brand \sqsubseteq \bot\}$, we get the probability $P(C') = Z^{-1}\exp(0)$. This time, we have that $C' \models \mathcal{C}^{\mathsf{D}}$ and C' is coherent, but since no weight axiom pair of $\mathcal{C}^{\mathsf{U}}$ is entailed in C' we have no weights w_c to sum up.*

- *If $C' = \{Cat \sqsubseteq Animal, Cat \sqcap Brand \sqsubseteq \bot, Jaguar \sqsubseteq Brand\}$, the probability is $P(C') = Z^{-1}\exp(1.2)$. Again, $\mathcal{C}^{\mathsf{D}}$ is entailed in C' and C' is coherent. Since one axiom $c = \langle Jaguar \sqsubseteq Brand, 1.2\rangle$ of the uncertain CBox $\mathcal{C}^{\mathsf{U}}$ is entailed in C', we take the exponential of its weight $w_c = 1.2$.*

- *If $C' = \{Cat \sqsubseteq Animal, Cat \sqcap Brand \sqsubseteq \bot, Jaguar \sqsubseteq Cat\}$, the probability is $P(C') = Z^{-1}\exp(1.4)$. Since $C' \models \mathcal{C}^{\mathsf{D}}$ and C' is coherent, the probability is not 0. This time, two axioms $c_1 = Jaguar \sqsubseteq Cat$ and $c_2 = Jaguar \sqsubseteq Animal$ are entailed in C' and are both elements of $\mathcal{C}^{\mathsf{U}}$, written as $c_1, c_2 \in \mathcal{C}^{\mathsf{U}}$. Axiom c_1 is explicitly encoded in C' while axiom c_2 can be inferred from $Jaguar \sqsubseteq Cat$ and $Cat \sqsubseteq Animal$. Consequently, we compute the sum over their corresponding weights $w_{c_1} = 0.5$ and $w_{c_2} = 0.9$.*

- *If $C' = \{Cat \sqsubseteq Animal, Cat \sqcap Brand \sqsubseteq \bot, Jaguar \sqsubseteq Cat, Jaguar \sqsubseteq Brand\}$, we get the probability $P(C') = 0$ because C' is incoherent. We refer the reader to Example 8 for an explanation.*

The normalization constant Z is calculated by summing up all (non-zero) probabilities of all possible CBoxes C'. However, even for this small example, the possible number of C' which are coherent and for which $C' \vDash \mathcal{C}^{\mathsf{D}}$ holds is not trivial to determine.

8.3 Towards a Markov Logic Representation

A way of efficient query answering in log-linear description logics is to translate its semantic into a Markov logic network. To that end, we represent CBoxes as sets of first-order sentences attached with weights modeling the uncertain and deterministic axioms. For this representation we ask the reader to recapitulate first-order logic and in particular the definitions of Herbrand base, Herbrand model, and Herbrand interpretation from Section 2.2. Furthermore, the reader should be familiar with Markov logic (see Section 2.3).

The transformation into a Markov logic network enables us to reuse the standard inference methods in ML. We can for instance compute the a-posteriori probability of description logic axioms by applying marginal inference queries. However, this is out of the scope of this thesis. We refer the interested reader to [NNS11] for details. In the following, we concentrate on maximum a-posteriori inference. If we apply the MAP query on the semantic of log-linear description logic we obtain a novel query type. This novel query computes the most probable *coherent* ontology. For solving this query we can apply the efficient solving techniques presented in Part I.

Transforming a description logic knowledge base for which consequence-driven reasoning is possible (see Section 8.1) into its log-linear variant and, finally, to its Markov logic representation follows five general steps. In the following sections, we dive deeper into each of these steps taking the description logic $\mathcal{EL}^{++}$ without nominals and concrete domains as an example.

First, the knowledge base has to be normalized to the set of axioms on which the completion rules are build. This means that composed axioms has to be correctly decomposed. In our case, we need to be able to know the set of normalized axioms for each non-normalized axiom after normalization. In many cases this is trivial. However, there are some exceptions like the domain restrictions in $\mathcal{EL}^{++}$. Section 8.3.1 introduces the normalization of an $\mathcal{EL}^{++}$ CBox.

After normalization of the axioms, Section 8.3.2 shows how the completion rules, which exist for any DL for which consequence driven-reasoning is possible, are transformed to first-order logic rules. Thereby, the normalized axioms are mapped to logical literals. Then, the uncertain axioms are integrated to the logical model in Section 8.3.3. This is were we need the link between the non-normalized and normalized axioms.

We are now in the position to define the novel query type of computing the most probable coherent ontology in Section 8.3.4. In Section 8.3.5 we finally combine all components and transform the problem to a specific Markov logic network. On

this ML network, the techniques for solving MAP queries in ML from Part I can be applied.

8.3.1 Normalization

The normalization step is a fundamental requirement for transforming a DL into a log-linear DL. Existing normalization algorithms have to be changed so that they are able to return the normalized axioms for *one* specific non-normalized axiom c. Usually, those algorithms normalize the whole CBox $\mathcal{C}$ without tracking the normalization for one specific axiom.

In this section, we focus on the normalization algorithms of $\mathcal{EL}^{++}$ as an example of one possible description logic. The presented normalization algorithm is combined from [BBL05a, BBL05b, BBL08] where they proved that all of the following transformations can be performed in linear time. In particular, we first introduce normalization leaving over range axioms and then point out how range axioms can be eliminated.

Let BC_c be the concept names in an axiom c. The normalization of an axiom c in $\mathcal{EL}^{++}$ with range restrictions but without concrete domains is done in two steps. First, we define the function $\mathsf{norm}_{\mathsf{A}}(c)$ which returns a set of axioms only containing the following normalized GCIs

$$\begin{aligned} &C_1 \sqsubseteq D;\\ &C_1 \sqsubseteq \exists r.C_2;\\ &C_1 \sqcap C_2 \sqsubseteq D;\\ &\exists r.C_1 \sqsubseteq D, \end{aligned}$$

the following normalized range restriction

$$\mathsf{ran}(r) \sqsubseteq C,$$

and/or the following normalized role inclusions axioms

$$\begin{aligned} &r \sqsubseteq s;\\ &r_1 \circ r_2 \sqsubseteq s \end{aligned}$$

where $C_1, C_2 \in \mathsf{BC}_c$ and $D \in \mathsf{BC}_c \cup \{\bot\}$.

The function $\mathsf{norm}_{\mathsf{A1}}(c)$ iteratively applies the normalization rules NF1-NF5 of Table 8.2 until none of the rules can be applied any more. Similarly the function $\mathsf{norm}_{\mathsf{A2}}(\mathcal{C})$ applies the rules NF6-NF8 iteratively on the set of axioms $\mathcal{C}$ until for axiom $c \in \mathcal{C}$ any new axiom can be inferred. The function $\mathsf{norm}_{\mathsf{A}}(c)$ is then defined as

$$\mathsf{norm}_{\mathsf{A}}(c) = \mathsf{norm}_{\mathsf{A2}}(\mathsf{norm}_{\mathsf{A1}}(c)).$$

The normalization rules in Table 8.2 are equivalent with the rules in [BBL05b], except that we added rule NF4 for normalizing range restrictions. Almost all rules follow some similar characteristics. The only exception is Rule NF8, where we split axioms of type $B \sqsubseteq C \sqcap D$ into $B \sqsubseteq C$ and $B \sqsubseteq D$. The basic idea of

NF1	$r_1 \circ \ldots \circ r_k \sqsubseteq s$	$\mapsto$	$\{r_1 \circ \ldots \circ r_{k-1} \sqsubseteq u, u \circ r_k \sqsubseteq s\}$
NF2A	$C \sqcap \hat{D} \sqsubseteq E$	$\mapsto$	$\{\hat{D} \sqsubseteq A, C \sqcap A \sqsubseteq E\}$
NF2B	$\hat{D} \sqcap C \sqsubseteq E$	$\mapsto$	$\{\hat{D} \sqsubseteq A, C \sqcap A \sqsubseteq E\}$
NF3	$\exists r.\hat{C} \sqsubseteq D$	$\mapsto$	$\{\hat{C} \sqsubseteq A, \exists r.A \sqsubseteq D\}$
NF4	$\mathsf{ran}(r) \sqsubseteq \hat{C}$	$\mapsto$	$\{\mathsf{ran}(r) \sqsubseteq A, A \sqsubseteq \hat{C}\}$
NF5	$\bot \sqsubseteq C$	$\mapsto$	$\varnothing$
NF6	$\hat{C} \sqsubseteq \hat{D}$	$\mapsto$	$\{\hat{C} \sqsubseteq A, A \sqsubseteq \hat{D}\}$
NF7	$B \sqsubseteq \exists r.\hat{C}$	$\mapsto$	$\{B \sqsubseteq \exists r.A, A \sqsubseteq \hat{C}\}$
NF8	$B \sqsubseteq C \sqcap D$	$\mapsto$	$\{B \sqsubseteq C, B \sqsubseteq D\}$

Table 8.2: The normalization rules derived from [BBL05b] without range elimination. All r_i, s are role names. All $\hat{C}, \hat{D} \notin \mathsf{BC}_c$ are the non-normalized axioms and u denotes a new role name and A a new concept name. All B, C, D and E can be already normalized or not yet.

all other rules is to introduce new concept names A like in rule NF2A-NF7 or new role names u like in rule NF1 in order to simplify axioms. The rules often focus on further simplifying a concept description $\hat{C}$ or $\hat{D}$ which is not normalized yet. It is important to know that the concepts B, C, D, and E are not necessarily normalized. In Rule NF2A the concept C can for example be in BC_c or not.

The reason for splitting the rule applications in two phases is to remain the linear size of the normalized CBox. If we apply all rules together, we might get a quadratic blowup due to the dublication of concept B by Rule NF8.

Currently, our normalization of an axiom c still contains axioms of type $\mathsf{ran}(r) \sqsubseteq C$. Eliminating these axiom types requires the normalized axioms of the whole CBox $\mathcal{C}$. A CBox $\mathcal{C}$ is transferred to a normalized CBox still containing range restrictions of the form $\mathsf{ran}(r) \sqsubseteq C$ where $C \in \mathsf{BC}_\mathcal{C}$ by normalizing each axiom c in $\mathcal{C}$ with

$$\mathsf{norm}_\mathsf{A}(\mathcal{C}) = \{\mathsf{norm}_\mathsf{A}(c) : c \in \mathcal{C}\}.$$

Example 23 illustrates the normalization function norm_A.

Example 23. *Our example ontology $\mathcal{O}_1$ from Example 7 is already normalized. Thus, we introduce a new ontology $\mathcal{O}_2$. Informally, this ontology describes the world from the eyes of a tiger.*

$Animal \sqcap Plant \sqsubseteq \bot$	*Something can not be an animal and a plant.*	*(1)*
$Tiger \sqsubseteq Animal \sqcap \exists eats.(Animal \sqcap \exists eats.Plant)$		*(2)*
	A Tiger is an animal which eats animals which eat plants.	
$\mathsf{ran}(eats) \sqsubseteq Animal$	*Everything which can be eaten is an animal.*	*(3)*

Axioms (1) and (3) are already normalized. Axiom (2) does not change when we apply the function $\mathsf{norm}_{\mathsf{A}1}$*, but for* $\mathsf{norm}_{\mathsf{A}2}$ *we obtain*

$\mathsf{norm}_{\mathsf{A2}}(Tiger \sqsubseteq Animal \sqcap \exists eats.(Animal \sqcap \exists eats.Plant)) =$
$\{Tiger \sqsubseteq Animal, Tiger \sqsubseteq \exists eats.A_1, A_1 \sqsubseteq Animal, A_1 \sqsubseteq \exists eats.Plant\}$

by applying rules NF8, NF6 introducing A_1 and NF8. Thus, the normalized ontology according to the function $\mathsf{norm}_{\mathsf{A}}$ *is the union of all normalized axioms:*

$\mathsf{norm}_{\mathsf{A}}(\mathcal{O}_2) =$
$\{Animal \sqcap Plant \sqsubseteq \bot,$
$Tiger \sqsubseteq Animal, Tiger \sqsubseteq \exists eats.A_1, A_1 \sqsubseteq Animal, A_1 \sqsubseteq \exists eats.Plant,$
$\mathsf{ran}(eats) \sqsubseteq Animal\}$

We are now in the position to define the function $\mathsf{norm}_{\mathsf{B}}(c, \mathcal{C}_{norm_A})$ which eliminates range restrictions of the form $\mathsf{ran}(r) \sqsubseteq C$. Algorithm 7 does specify the exact steps of the function. The input is a normalized CBox $\mathcal{C}_{norm_A} = \mathsf{norm}_{\mathsf{A}}(\mathcal{C})$ and a description logic axiom $c \in \mathcal{C}_{norm_A}$. It outputs the set of normalized axioms for c. However, it also changes the input set $\mathcal{C}_{norm_A}$ such that all range restriction axioms are correctly normalized if the function $\mathsf{norm}_{\mathsf{B}}(c, \mathcal{C}_{norm_A})$ has been called for every $c \in \mathcal{C}_{norm_A}$.

The only axiom types which interact with range restrictions are of form $c \notin \mathsf{ran}(r) \sqsubseteq A$ or $c \notin C \sqsubseteq \exists s.D$. If the input axiom c does not have this form, c is already normalized and is returned (Line 1). Else, we initialize the return value $norm$, which will contain the normalized axioms for c at the end, with the empty set (Line 2).

Then, we iterate over each s for which $\mathcal{C}$ contains role inclusions $r \sqsubseteq t_1, t_1 \sqsubseteq t_2, \ldots, t_n \sqsubseteq s$ (Line 3). In case the input axiom c has the form $\mathsf{ran}(s) \sqsubseteq A$ (Line 4), we

- delete each axiom $\mathsf{ran}(s) \sqsubseteq A$ in $\mathcal{C}_{norm_A}$ (Line 5).
- introduce a new concept name $X_{s,D}$ for each axiom $C \sqsubseteq \exists s.D$ in CBox $\mathcal{C}_{norm_A}$ (Line 7). Intuitively, $X_{s,D}$ denotes the range of s intersected with the extension of concept name D.
- exchange every axiom of the form $C \sqsubseteq \exists s.D$ in CBox $\mathcal{C}_{norm_A}$ with the axioms $C \sqsubseteq \exists s.X_{s,D}$, $X_{s,D} \sqsubseteq D$, and $X_{s,D} \sqsubseteq A$ (Lines 8-9) and add $X_{s,D} \sqsubseteq A$ to the return value $norm$ (Line 10).
- if s is defined as a reflexive role with $\epsilon \sqsubseteq s$ in $\mathcal{C}_{norm_A}$ (Line 12), then add $\top \sqsubseteq A$ to $\mathcal{C}_{norm_A}$ (Line 13) and to return value $norm$ (Line 14).
- return the axioms in set $norm$ (Line 22).

If axiom c has the form $C \sqsubseteq \exists s.D$ (Line 16) and if a new concept name $X_{s,D}$ has been introduced due to a range restriction (Line 17), return the set of axioms $\{C \sqsubseteq \exists s.X_{s,E}, X_{s,E} \sqsubseteq D\}$ (Line 18 and Line 22).

When we compare this algorithm for eliminating range restrictions with the original algorithm from [BBL08], we notice two differences. First, our algorithm

Algorithm 7 Algorithm for range restriction normalization (inspired by [BBL08]).

Input: $\mathcal{C}_{norm_A}$: a normalized CBox containing range restrictions $\mathsf{norm_A}(\mathcal{C})$
Input: c: description logic axiom where $c \in \mathcal{C}_{norm_A}$
Output: $norm$: set of normalized axioms
$\mathsf{norm_B}(c, \mathcal{C}_{norm})$

if not $c \equiv \mathsf{ran}(r) \sqsubseteq A$ **and not** $c \equiv C \sqsubseteq \exists s.D$ **then return** $\{c\}$ **end if**
$norm \leftarrow \varnothing$
for each s for which $\mathcal{C}$ contains role inclusions $r \sqsubseteq t_1, t_1 \sqsubseteq t_2, \ldots, t_n \sqsubseteq s$ **do**
  if $c \equiv \mathsf{ran}(s) \sqsubseteq A$ **then**
    delete axiom $\mathsf{ran}(s) \sqsubseteq A$ in $\mathcal{C}_{norm_A}$.
    for each $C \sqsubseteq \exists s.D$ in CBox $\mathcal{C}_{norm_A}$ **do**
      introduce a new concept name $X_{s,D}$.
      delete $C \sqsubseteq \exists s.D$ from $\mathcal{C}_{norm_A}$.
      add $C \sqsubseteq \exists s.X_{s,D}$, $X_{s,D} \sqsubseteq D$, and $X_{s,D} \sqsubseteq A$ to $\mathcal{C}_{norm_A}$.
      add $X_{s,D} \sqsubseteq A$ to $norm$.
    end for
    if s is defined as a reflexive role with $\epsilon \sqsubseteq s$ in $\mathcal{C}_{norm_A}$ **then**
      add $\top \sqsubseteq A$ to $\mathcal{C}_{norm_A}$.
      add $\top \sqsubseteq A$ to $norm$.
    end if
  else if $C \sqsubseteq \exists s.D$ **then**
    if a new concept name $X_{s,D}$ has been introduced due to a range restriction **then**
      add $C \sqsubseteq \exists s.X_{s,E}$ and $X_{s,E} \sqsubseteq D$ to $norm$.
    end if
  end if
end for
return $norm$

(1)	$\mathsf{norm}(Animal \sqcap Plant \sqsubseteq \bot) = \{Animal \sqcap Plant \sqsubseteq \bot\}$
(2)	$\mathsf{norm}(Tiger \sqsubseteq Animal \sqcap \exists eats.(Animal \sqcap \exists eats.Plant)) =$ $\{Tiger \sqsubseteq Animal, Tiger \sqsubseteq \exists eats.X_{eats,A_1}, X_{eats,A_1} \sqsubseteq A_1,$ $A_1 \sqsubseteq Animal, A_1 \sqsubseteq \exists eats.X_{eats,Plant}, X_{eats,Plant} \sqsubseteq Plant\}$
(3)	$\mathsf{norm}(\mathsf{ran}(eats) \sqsubseteq Animal) =$ $X_{eats,A_1} \sqsubseteq Animal, X_{eats,Plant} \sqsubseteq Animal.$

Table 8.3: Normalized axioms of Ontology $\mathcal{O}_2$ of Example 23.

returns the axioms for one specific range axiom c in step (4) while the algorithm in [BBL08] just returns the changed CBox. Second, we avoid the definition of $A \in ran_{\mathcal{C}}(r)$ of [BBL08] by explicitly performing the changes for every s for which $r \sqsubseteq t_1, \ldots, t_n \sqsubseteq s \in \mathcal{C}$. Of course, this includes the case $s \equiv r$. These changes do not influence soundness and completeness of the normalization.

In the last step, we only return those axioms, which are added due to the normalization of axiom c. From Step (3) we only add the axioms of type $X_{s,E} \sqsubseteq A$ because the axioms $C \sqsubseteq \exists s.X_{s,E}$ and $X_{s,E} \sqsubseteq D$ are equivalent to the axiom $C \sqsubseteq \exists s.D$. This latter axiom is not added due to the axiom c but has been in the CBox before. In Step (4) we add $\top \sqsubseteq A$, because this axiom is directly derived from the axiom c.

Finally, we combine both normalization functions $\mathsf{norm_A}$ and $\mathsf{norm_B}$ and are now able to normalize an axiom c with respect to a CBox $\mathcal{C}$ with the function

$$\mathsf{norm}(c) = \{\mathsf{norm_B}(d, \mathsf{norm_A}(\mathcal{C})) : d \in \mathsf{norm_A}(c)\}.$$

This function returns the set of normalized axioms of c by first applying $\mathsf{norm_A}$ on c and, then, eliminating the range restrictions with $\mathsf{norm_B}$ on each axiom d returned by $\mathsf{norm_A}$. We result in the full normalization of a CBox $\mathcal{C}$ written as $\mathsf{norm}(\mathcal{C})$, when we normalize each axiom $c \in \mathcal{C}$ with $\mathsf{norm}(c)$. Example 24 illustrates the application of the function $\mathsf{norm_B}$.

Example 24. *We recapitulate ontology $\mathcal{O}_2$ and its normalized version $\mathsf{norm_A}(\mathcal{O}_2)$ from the previous example (Example 23). This example examines the effect of the function $\mathsf{norm_B}$ on the range axiom $\mathsf{ran}(eats) \sqsubseteq Animal$. It has the form $\mathsf{ran}(r) \sqsubseteq A$ where axiom A equals $Animal$ and role $r = s = eats$. Here, only one role is affected because we do not have any role inclusions. According to Algorithm 7, we*

- *delete $\mathsf{ran}(eats) \sqsubseteq Animal$ from $\mathsf{norm_A}(\mathcal{O}_2)$ (Line 5),*
- *introduce new concept names X_{eats,A_1} for $Tiger \sqsubseteq \exists eats.A_1$ and $X_{eats,Plant}$ for $A_1 \sqsubseteq \exists eats.Plant$ (Line 7),*
- *exchange the axioms $Tiger \sqsubseteq \exists eats.A_1$ with $Tiger \sqsubseteq \exists eats.X_{eats,A_1}$, $X_{eats,A_1} \sqsubseteq A_1$, and $X_{eats,A_1} \sqsubseteq Animal$ as well as $A_1 \sqsubseteq \exists eats.Plant$ with $A_1 \sqsubseteq \exists eats.X_{eats,Plant}$, $X_{eats,Plant} \sqsubseteq Plant$, and $X_{eats,Plant} \sqsubseteq Tiger$ (Lines 8-9). Furthermore, we add the description logic axioms $\{X_{eats,A_1} \sqsubseteq Animal, X_{eats,Plant} \sqsubseteq Animal\}$ to $norm$ (Line 10).*

- *add no further axioms, since no reflexive roles are defined in our example (Line 12-14), and*
- *return* $norm$ *(Line 22).*

For the axiom $Tiger \sqsubseteq \exists eats.A_1$ *the function* $\mathsf{norm_B}$ *returns the set* $\{Tiger \sqsubseteq \exists eats.X_{eats,A_1}, X_{eats,A_1} \sqsubseteq A_1\}$ *due to Lines 16-18 and Line 22. All other axioms are not changed, because they are neither of form* $C \sqsubseteq \exists s.D$ *nor of form* $\mathsf{ran}(r) \sqsubseteq A$*. We now conclude this example with Table 8.3 which contrasts the non-normalized axioms with the normalized axioms.*

Most normalization algorithms like the ones described by [BBL05b, BBL08] just perform the normalization for the whole CBox $\mathcal{C}$. However, we require to know the mapping of one specific axiom c to its normalized axioms. In most cases it is trivial to receive this mapping. However, in some cases it requires caution like in the range elimination algorithm of $\mathcal{EL}^{++}$.

8.3.2 Transformation of Completion Rules to First Order Logic

The second step of translating a logic to its corresponding log-linear logic is to translate the completion rules into a set of deterministic first-order formulas. Each description logic for which consequence-driven reasoning is possible, has a finite set of completion rules. We refer the reader to Section 8.1 for a list of those description logics. In the case of $\mathcal{EL}^{++}$ Baader et al. [BBL05a] build them on the basis of the six different axiom types occurring in an normalized CBox and prove their soundness and completeness.

Let $\mathcal{C}$ be the implicitly defined initial CBox which contains the normalized axioms. Let $\mathcal{BC}$ be the set of all concept names. Let $C_i \in \mathcal{BC} \cup \{\bot\}$ be concept names including the bottom element $\bot$, and r_i be role names. According to [BBL05a], we can classify the ontology by deriving new axioms of the types $C_1 \sqsubseteq D$ and $C_1 \sqsubseteq \exists r.C_2$. To that end, we apply the rules in Table 8.4 iteratively until all rules do not derive new axioms.

The general structure of the rules in Table 8.4 are all similar. First, they iterate over all combinations of a certain number of concepts or roles. Please note that those concepts and roles are not mutually exclusive. Thus, it might occur in some assignments that e.g. $C_1 = C_2$. Then, we define the axiom which is derived from the current CBox $\mathcal{C}$ followed by the conditions which must hold to derive the axiom. We only add axioms which are either of the form $C_1 \sqsubseteq C_2$ (added by rules CR1, CR2, CR4, and CR5) or of the form $C_1 \sqsubseteq \exists r.C_2$ (added by rules CR3, CR10, and CR11). All conditions but the last one state the axioms which have to be in CBox $\mathcal{C}$ and from which the axiom that we want to add is derived. In rule CR1 we formulate for example the transitivity characteristic of the subsumption axiom which says that if an axiom C_1 is subsumed by an axiom C_2 and the axiom C_2 is again subsumed by an axiom C_3, we can infer that the axiom C_1 is also subsumed by the axiom C_3. The last condition always requires that the CBox $\mathcal{C}$ must

CR1	For all C_1, C_2, C_3 we add $C_1 \sqsubseteq C_3$ to $\mathcal{C}$ if $C_1 \sqsubseteq C_2 \in \mathcal{C}$, $C_2 \sqsubseteq C_3 \in \mathcal{C}$, and $C_1 \sqsubseteq C_3 \notin \mathcal{C}$
CR2	For all C_1, C_2, C_3, C_4 we add $C_1 \sqsubseteq C_4$ to $\mathcal{C}$ if $C_1 \sqsubseteq C_2 \in \mathcal{C}$, $C_1 \sqsubseteq C_3 \in \mathcal{C}$, and $C_2 \sqcap C_3 \sqsubseteq C_4 \notin \mathcal{C}$
CR3	For all C_1, C_2, C_3 and all r we add $C_1 \sqsubseteq \exists r.C_3$ to $\mathcal{C}$ if $C_1 \sqsubseteq C_2 \in \mathcal{C}$, $C_2 \sqsubseteq \exists r.C_3 \in \mathcal{C}$, and $C_1 \sqsubseteq \exists r.C_3 \notin \mathcal{C}$
CR4	For all C_1, C_2, C_3, C_4 and all r we add $C_1 \sqsubseteq C_4$ to $\mathcal{C}$ if $C_1 \sqsubseteq \exists r.C_2 \in \mathcal{C}$, $C_2 \sqsubseteq C_3 \in \mathcal{C}$, $\exists r.C_3 \sqsubseteq C_4 \in \mathcal{C}$, and $C_1 \sqsubseteq C_4 \notin \mathcal{C}$
CR5	For all C_1, C_2 and all r we add $C_1 \sqsubseteq \bot$ to $\mathcal{C}$ if $C_1 \sqsubseteq \exists r.C_2 \in \mathcal{C}$, $C_2 \sqsubseteq \bot \in \mathcal{C}$, and $C_1 \sqsubseteq \bot \notin \mathcal{C}$
	... We exclude the completion rules CR6-CR9 for nominals and concrete domains ...
CR10	For all C_1, C_2 and all r_1, r_2 we add $C_1 \sqsubseteq \exists r_3.C_3$ to $\mathcal{C}$ if $C_1 \sqsubseteq \exists r_1.C_2 \in \mathcal{C}$, $C_2 \sqsubseteq \exists r_2.C_3 \in \mathcal{C}$, and $C_1 \sqsubseteq \exists r_3.C_3 \notin \mathcal{C}$
CR11	For all C_1, C_2, C_3 and all r_1, r_2, r_3 we add $C_1 \sqsubseteq \exists r_1.C_3$ to $\mathcal{C}$ if $r_1 \sqsubseteq r_2 \in \mathcal{C}$, $C_1 \sqsubseteq \exists r_2.C_2 \in \mathcal{C}$, and $C_1 \sqsubseteq \exists r_1.C_2 \notin \mathcal{C}$

Table 8.4: The completion rules from [BBL05a] (from [NNS13]). All C_i are concept names, and all r_i are role names.

not contain the the axiom already. We exclude rules CR6-CR9 since we exclude nominals and concrete domains. Please note that our representation slightly differs from the representation in [BBL05a] since our representation makes the following translation to first-order rules more obvious.

We now turn our attention to the transformation of those rules to first-order logic formulas. In this translation we introduce a first-order literal for each of the axiom types which occur in the transformation rules. Furthermore, each completion rule is translated into one first-order rule, which is essentially the same as a hard Markov logic rule.

In case of the description logic $\mathcal{EL}^{++}$ we define first-order literals fore each of the six axiom types occurring in a normalized CBox. We are now in the position to define a bijective function φ that, given a finite set of concept and role names $\mathsf{N_U}$, maps each *normalized* $\mathcal{EL}^{++}$ CBox over $\mathsf{N_U}$ to a subset of the Herbrand base of $\mathcal{F}$ with respect to $\mathsf{N_U}$.

Definition 4 (CBox Mapping)**.** *Let* $\mathsf{N_C}$ *and* $\mathsf{N_R}$ *be sets of concept and role names and let* $\mathsf{N_U} \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ *be a finite set. Let* $\mathcal{T}$ *be the set of normalized* $\mathcal{EL}^{++}$ *axioms constructible from* $\mathsf{N_U}$*. Moreover, let* $\mathcal{H}$ *be the Herbrand base of* $\mathcal{F}$ *with respect to* $\mathsf{N_U}$*. The function* $\varphi : \wp(\mathcal{T}) \to \wp(\mathcal{H})$ *maps normalized CBoxes to subsets of* $\mathcal{H}$ *as*

F_1	$\forall c : sub(c, c)$
F_2	$\forall c : sub(c, \top)$
F_3	$\forall c_1, c_2, c_3 : sub(c_1, c_2) \wedge sub(c_2, c_3) \Rightarrow sub(c_1, c_3)$
F_4	$\forall c_1, c_2, c_3, c_4 : sub(c_1, c_2) \wedge sub(c_1, c_3) \wedge int(c_2, c_3, c_4) \Rightarrow sub(c_1, c_4)$
F_5	$\forall c_1, c_2, c_3, r : sub(c_1, c_2) \wedge rsup(c_2, r, c_3) \Rightarrow rsup(c_1, r, c_3)$
F_6	$\forall c_1, r, c_2, c_3, c_4 : rsup(c_1, r, c_2) \wedge sub(c_2, c_3) \wedge rsub(c_3, r, c_4) \Rightarrow sub(c_1, c_4)$
F_7	$\forall c_1, c_2, r_1, r_2 : rsup(c_1, r_1, c_2) \wedge psub(r_1, r_2) \Rightarrow rsup(c_1, r_2, c_2)$
F_8	$\forall c_1, c_2, c_3, r_1, r_2, r_3 : rsup(c_1, r_1, c_2) \wedge rsup(c_2, r_2, c_3) \wedge pcom(r_1, r_2, r_3) \Rightarrow rsup(c_1, r_3, c_3)$
F_9	$\forall c : \neg sub(c, \bot)$

Table 8.5: The set of first-order formulas $\mathcal{F}$ [NNS11]. Groundings of the formulas have to be compatible with the types of the predicates specified in Definition 4. $\bot$ and $\top$ are constant symbols representing the bottom and top concept.

follows: $(\varphi(\mathcal{C}) = \bigcup_{c \in \mathcal{C}} \varphi(c))$

$$
\begin{array}{lcl}
C_1 \sqsubseteq D & \mapsto & sub(C_1, D) \\
C_1 \sqcap C_2 \sqsubseteq D & \mapsto & int(C_1, C_2, D) \\
C_1 \sqsubseteq \exists r.C_2 & \mapsto & rsup(C_1, r, C_2) \\
\exists r.C_1 \sqsubseteq D & \mapsto & rsub(C_1, r, D) \\
r \sqsubseteq s & \mapsto & psub(r, s) \\
r_1 \circ r_2 \sqsubseteq r_3 & \mapsto & pcom(r_1, r_2, r_3).
\end{array}
$$

All predicates are typed meaning that $r, s, r_i (1 \leq i \leq 3)$ are role names, C_1, C_2 are basic concept descriptions, and D is a basic concept description or the bottom concept.

The set of formulas $\mathcal{F}$ are listed in Table 8.5. The constant symbols $\bot$ and $\top$ represent the bottom and top concept. The formulas are derived from the $\mathcal{EL}^{++}$ completion rules of Table 8.4. In the formulas F_1 and F_2 we add the trivial axioms $C \sqsubseteq C$ and $C \sqsubseteq \top$ for all concepts C. Please note that these two rules are encompassed by rule $F3$. The formulas F_3, F_4, F_5, F_6, F_7 and F_8 directly map to the derivation rules $CR1, CR2, CR3, CR4, CR10$, and $CR11$, respectively. Formula F_9 ensures coherency since it prohibits the derivation of axioms of the form $C \sqsubseteq \bot$.

We can omit the derivation rule $CR5$, which would result in the first-order formula $\forall c_1, c_2, r : rsup(c_1, r, c_2) \wedge sub(c_2, \bot) \Rightarrow sub(c_1, \bot)$, because this formula never infers new information. This is the case because $sub(c_2, \bot)$ is always false because formula F_9 states that $\forall c : \neg sub(c, \bot)$. Due to better readability, we decided to write the formulas as implications. Please note that implications can be transformed to clauses as introduced in Section 2.2 by negating the left part of the implication.

We now prove that, relative to a finite set of concept and role names, the function φ induces a one-to-one correspondence between Herbrand models of the first-order theory $\mathcal{F}$ and coherent and classified $\mathcal{EL}^{++}$ CBoxes.

Lemma 5. *Let* $\mathsf{N_C}$ *and* $\mathsf{N_R}$ *be sets of concept and role names and let* $\mathsf{N_U} \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ *be a finite set. Let* $\mathcal{T}$ *be the set of normalized* $\mathcal{EL}^{++}$ *axioms constructible from* $\mathsf{N_U}$ *and let* $\mathcal{H}$ *be the Herbrand base of* $\mathcal{F}$ *with respect to* $\mathsf{N_U}$*. Then,*

(a) for any $\mathcal{C} \subseteq \mathcal{T}$ *we have that if* $\mathcal{C}$ *is classified and coherent then* $\varphi(\mathcal{C})$ *is a Herbrand model of* $\mathcal{F}$*; and*

(b) for any $H \subseteq \mathcal{H}$ *we have that if* H *is a Herbrand model of* $\mathcal{F}$ *then* $\varphi^{-1}(H)$ *is a classified and coherent CBox.*

Proof. In this proof, we utilize the fact that the completion rules in Table 8.4 are proven to classify $\mathcal{EL}^{++}$ knowledge bases without nominals and concrete domains [BBL05a]. These set of rules are sound and complete. Additionally, we use the fact that each completion rule CRX has a direct translation to a first-order rule $F_y \in \mathcal{F}$ (of Table 8.5) where CR1, CR2, CR3, CR4, CR10, and CR11 map to rules F_3, F_4, F_5, F_6, F_7, and F_8, respectively. Rule F_1 is a special case of CR1 where $\varphi^{-1}(c) = C_1 = C_2 = C_3$ and rule F_2 is a special case of CR1 where $\varphi^{-1}(c) = C_1 = C_2$ and $\varphi^{-1}(\top) = C_3$.

(a) Let us assume that $\varphi(\mathcal{C})$ is no Herbrand model and $\mathcal{C}$ is classified and coherent. Then, there must exist at least one literal $\ell \in \varphi(\mathcal{C})$ such that one first-order formula $F_y \in \mathcal{F}$ is violated. If F_9 is violated, we conclude that $\ell \equiv sub(C, \bot)$. Since $C \sqsubseteq \bot = \varphi^{-1}(sub(C, \bot))$ we conclude that $\mathcal{C}$ is not coherent which is a contradiction.

If any other formula F_y is violated, we conclude that the corresponding completion rule CRX derives $\varphi^{-1}(\ell) \notin \mathcal{C}$. Since the set of completion rules is proven to be sound for $\mathcal{EL}^{++}$ knowledge bases without nominals and concrete domains [BBL05a], we conclude that $\mathcal{C}$ is not classified. This again results in a contradiction.

(b) Let us assume that $\varphi^{-1}(H)$ is not classified but H is a Herbrand model. Then, there exist at least one axiom $c \notin \varphi^{-1}(H)$ which can be derived with one of the completion rules since it is proven that they are complete for $\mathcal{EL}^{++}$ knowledge bases without nominals and concrete domains [BBL05a]. Let us refer to this completion rule with CRX. Then, the first-order rule F_y must derive from H the corresponding axiom $\varphi(c) \notin H$. Thus, H is no Herbrand model of $\mathcal{F}$ since rule F_y is violated. This is a contradiction.

Additionally, let us assume that $\varphi^{-1}(H)$ is not coherent but H is a Herbrand model. Then, there exists according to the definition of incoherence a class $C \in \varphi^{-1}(H)$ such that $\{C \sqsubseteq \bot\} \in \varphi^{-1}(H)$. From this we conclude, that $\varphi(C \sqsubseteq \bot) \in H$. However, this leads to a violation of rule F_9 and thus to the conclusion that H is no Herbrand model. Again, this is a contradiction.

□

From Lemma 5 we know that, relative to a finite set $\mathsf{N_U}$ of concept and role names, each normalized CBox over $\mathsf{N_U}$ that is classified and coherent, corresponds to exactly one Herbrand model of $\mathcal{F}$. Thus, we can use the first-order logic formulas in Table 8.5 to classify a $\mathcal{EL}^{++}$ CBox and to check if this CBox is coherent.

Example 25 illustrates how the translation is performed and how the incoherency of a simple ontology can be derived. We encourage the reader to compare the method of determining the incoherency via translating the ontology to first-order clauses with the traditional method illustrated in Example 8.

Example 25. *Let $\mathcal{O}_1$ be the ontology which we defined in Example 7. The ontology has the concept names $\mathsf{N_C} = \{Animal, Brand, Cat, Jaguar\}$ but no role names. According to Definition 4 we can transform the axioms of the ontology to first-order literals as follows:*

$Cat \sqsubseteq Animal \quad \mapsto \quad sub(Cat, Animal)$
$Cat \sqcap Brand \sqsubseteq \bot \quad \mapsto \quad int(Cat, Brand, \bot)$
$Jaguar \sqsubseteq Cat \quad \mapsto \quad sub(Jaguar, Cat)$
$Jaguar \sqsubseteq Brand \quad \mapsto \quad sub(Jaguar, Brand)$

If we now apply the set of first-order formulas $\mathcal{F}$ from Table 8.5 we can derive the following new axioms with the following formulas:

Formula	*Axioms*
F_1	$sub(Animal, Animal), sub(Brand, Brand), sub(Cat, Cat),$ $sub(Jaguar, Jaguar)$
F_2	$sub(Animal, \top), sub(Brand, \top), sub(Cat, \top), sub(Jaguar, \top)$
F_3	$sub(Jaguar, Animal)$
F_4	$sub(Jaguar, \bot)$
F_9	$\neg sub(Animal, \bot), \neg sub(Brand, \bot), \neg sub(Cat, \bot), \neg sub(Jaguar, \bot)$

The axioms inferred from Formulas F_1 and F_2 are trivial. Formula F_3 infers that $sub(Jaguar, Animal)$ because

$$sub(Jaguar, Cat) \wedge sub(Cat, Animal) \Rightarrow sub(Jaguar, Animal)$$

and Formula F_4 infers that $sub(Jaguar, \bot)$ due to

$$sub(Jaguar, Cat) \wedge sub(Jaguar, Brand) \wedge int(Cat, Brand, \bot) \Rightarrow sub(Jaguar, \bot).$$

Formulas F_5 to F_8 are not applied since our model does not contain role names.

Due to the existence of Formula F_9 the grounded set of first-order formulas is no model since Formula F_9 does not allow $sub(Jaguar, \bot)$ to be true. However, this fact was inferred by Formula F_4. According to Lemma 5 we can conclude that our original CBox is not coherent.

If the ontology is coherent, our rule set classifies the ontology. However, if the ontology is incoherent the rules do not guarantee full classification. An incoherent

ontology is always not completely classified, if the corresponding first-order logic variant of the completion rule $CR5$ can infer new knowledge. If we add the first-order logic variant of $CR5$ to our rule set, incoherent ontologies would also be classified. In our case, however, it is enough that we can guarantee the ontology to be classified if it is coherent.

8.3.3 Addition of Axioms

Until now we translated the coherency checking and the classification into a set of first-order formulas and translated the description logic axioms into first-order logic literals. Speaking in terms of Markov logic networks, our network just contains hard formulas at the moment. The next step is to add the uncertain axioms from $\mathcal{C}^{\mathsf{U}}$ and the certain axioms from $\mathcal{C}^{\mathsf{D}}$ to the model.

We extend the normalization of $\mathcal{EL}^{++}$ CBoxes to log-linear $\mathcal{EL}^{++}$ CBoxes as follows. The log-linear CBox is called $\mathcal{EL}^{++}_{\mathsf{LL}}$.

Definition 6. *Let* $\mathcal{C} = (\mathcal{C}^{\mathsf{D}}, \mathcal{C}^{\mathsf{U}})$ *be a* $\mathcal{EL}^{++}_{\mathsf{LL}}$ *CBox. Then,* $\mathsf{norm}_{\mathsf{LL}}(\mathcal{C}) = \mathsf{norm}(\mathcal{C}^{\mathsf{D}}) \cup \bigcup_{\langle c, w_c\rangle \in \mathcal{C}^{\mathsf{U}}} \mathsf{norm}(\{c\})$.

In this definition we define the normalized log-linear CBox $\mathsf{norm}_{\mathsf{LL}}(\mathcal{C})$ out of the non-normalized CBox $\mathcal{C}$ which consists of the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ and the uncertain CBox $\mathcal{C}^{\mathsf{U}}$. The normalization function $\mathsf{norm}(\ldots)$ has already been introduced in Section 8.3.1. In particular, the function $\mathsf{norm}(\{c\})$ returns the normalized axioms for one single axiom c.

The normalized log-linear CBox $\mathsf{norm}_{\mathsf{LL}}(\mathcal{C})$ consists of all distinct elements of all normalized axioms within the deterministic CBox $\mathsf{norm}(\mathcal{C}^{\mathsf{D}})$ and of the union of all axioms c in the uncertain CBox $\bigcup_{(c,w_c)\in\mathcal{C}^{\mathsf{U}}} \mathsf{norm}(\{c\})$. We need this definition of a normalized log-linear CBox later, when we formally define the computation of the maximum a-posteriori (MAP) query.

Lemma 5 provides the justification for constructing the logical representation of a $\mathcal{EL}^{++}_{\mathsf{LL}}$ CBox as follows:

Let $\mathcal{G}$ be a set of *weighted* ground formulas carrying the uncertain information. This set is derived from the axioms in the uncertain CBox $\mathcal{C}^{\mathsf{U}}$ as follows. For every pair $\langle c, w_c\rangle \in \mathcal{C}^{\mathsf{U}}$ we add the conjunction of ground atoms

$$\langle \bigwedge_{\mathsf{g}\in\varphi(\mathsf{norm}(\{c\}))} \mathsf{g}, w_c\rangle$$

to $\mathcal{G}$. For every axiom $c \in \mathcal{C}^{\mathsf{U}}$ we retrieve its set of axioms $\mathsf{norm}(\{c\})$. This set contains at least one axiom. Those normalized axioms are then transformed to its logical equivalences as described in Definition 4. The conjunction of these equivalences are then put as one weighted ground formula into the set of $\mathcal{G}$. The number of weighted ground formulas in $\mathcal{G}$ equals the number of weighted non-normalized axioms in $\mathcal{C}^{\mathsf{U}}$.

The set $\mathcal{K}$ is constructed analogously from the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ except that we do not associate weights with the ground formulas.

$$\mathcal{K} = \{\mathsf{g} : \mathsf{g} \in \varphi(\mathsf{norm}(\{c\})), \forall c \in \mathcal{C}^{\mathsf{D}}\}$$

Like before, we go through all axioms c in $\mathcal{C}^{\mathsf{D}}$ and fetch the logical ground atoms of the normalized axiom c with $\varphi(\mathsf{norm}(\{c\}))$. The difference of the construction of set $\mathcal{K}$ compared to the construction of set $\mathcal{G}$ lies in the fact that the information from which axiom c the ground atom g is derived is not longer maintained in set $\mathcal{K}$. Thus, the size of the set $\mathcal{K}$ is smaller or equal than the number of axioms in the deterministic CBox $\mathcal{C}^{\mathsf{D}}$.

Intuitively, this difference can be explained as follows: Later in the translation to the Markov logic network, the ground atoms in set $\mathcal{K}$ are *just* evidence, where each atom g is set to true, irrespectively from which axiom c it was derived. However, the formulas in set $\mathcal{G}$ are translated to weighted soft formulas.

With these two transformations, we bridged the gap between the ontological perspective and the logical perspective, which can be seen as the Markov logic perspective. The axioms in the uncertain CBox $\mathcal{C}^{\mathsf{U}}$ has been normalized and transformed to its logical equivalences stored in set $\mathcal{G}$. The same was done for the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ where the logical equivalences of the normalized axioms are put into the set $\mathcal{K}$.

We illustrate this transformation again on an example. Since our log-linear ontology $\mathcal{O}_1^{LL}$ from Example 21 is already normalized and thus would result in no conjunctions of soft weights, we define a new log-linear variant of $\mathcal{O}_2$ from Example 23 and illustrate how it is transferred to the logical perspective in Example 26.

Example 26. *We assign weights to some of the axioms of ontology $\mathcal{O}_2$ from Example 23 and create the log-linear ontology $\mathcal{O}_2^{LL}$:*

	$Animal \sqcap Plant \sqsubseteq \bot$	*(1)*
$\mathcal{C}^{\mathsf{D}}$	$\langle Tiger \sqsubseteq Animal \sqcap \exists eats.(Animal \sqcap \exists eats.Plant), 1.1\rangle$	*(2)*
$\mathcal{C}^{\mathsf{U}}$	$\langle \mathsf{ran}(eats) \sqsubseteq Animal, 0.3\rangle$	*(3)*

Table 8.3 contains the mapping of the normalized axioms from $\mathcal{O}_2$. From these, we can derive the normalized ontology

$$\begin{aligned}
&\mathsf{norm}_{\mathsf{LL}}(\mathcal{O}_2^{LL}) = \mathsf{norm}(\mathcal{C}^{\mathsf{D}}) \cup \textstyle\bigcup_{\langle c, w_c\rangle \in \mathcal{C}^{\mathsf{U}}} \mathsf{norm}(\{c\}) = \\
&\{Animal \sqcap Plant \sqsubseteq \bot\} \cup \\
&\{Tiger \sqsubseteq Animal, Tiger \sqsubseteq \exists eats.X_{eats,A_1}, X_{eats,A_1} \sqsubseteq A_1, \\
&A_1 \sqsubseteq Animal, A_1 \sqsubseteq \exists eats.X_{eats,Plant}, X_{eats,Plant} \sqsubseteq Plant \\
&X_{eats,A_1} \sqsubseteq Animal, X_{eats,Plant} \sqsubseteq Animal\}.
\end{aligned}$$

The set $\mathcal{K}$ containing the logical axioms for the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ and the set $\mathcal{G}$ containing the weight axiom pairs for the uncertain CBox $\mathcal{C}^{\mathsf{U}}$ consist of:

$\mathcal{K}$	$int(Animal, Plant, \bot)$	*(1)*
$\mathcal{G}$	$\langle sub(Tiger, Animal) \wedge rsup(Tiger, eats, X_{eats,A_1}) \wedge$ $sub(X_{eats,A_1}, A_1) \wedge sub(A_1, Animal) \wedge rsup(A_1, eats, X_{eats,Plant}) \wedge$ $sub(X_{eats,Plant}, Plant), 1.1\rangle$	*(2)*
	$\langle sub(X_{eats,A_1}, Animal) \wedge sub(X_{eats,Plant}, Animal), 0.3\rangle$	*(3)*

Please refer to Table 8.3 for the full set of normalized axioms.

8.3.4 The Most Probable Coherent Ontology Query

Under the given syntax and semantics of log-linear description logics we can ask the same two types of queries as in Markov networks or Markov logic. We distinguish between the marginal inference introduced in Section 2.1.2 and the maximum a-posteriori inference defined in Section 2.1.3. In context of log-linear description logics, the conditional probability query, which is in principle a more specific version of the marginal query, answers the questions: *"Given a log-linear description logic CBox, what is the probability of a conjunction of axioms?"*. The difference compared to the traditional conditional probability query lies in the fact, that we condition on the axioms in the deterministic CBox. For details and for an algorithm solving conditional probability queries we refer the reader to [NNS11].

In this thesis we focus on the maximum a-posteriori (MAP) query. In case of log-linear description logic the query answers the question: *"Given a log-linear $\mathcal{EL}^{++}$ CBox, what is a most probable coherent deterministic $\mathcal{EL}^{++}$ CBox over the same concept and role names?"* In the context of uncertain description logics, the MAP query is crucial as it infers a most probable classical ontology from an uncertain one. The MAP query also captures two important problems that frequently occur in the context of Ontology learning and ontology matching. Recently, the MAP query of log-linear description logics have also been used in activity recognition. We refer the reader to Section 10.2 for details of these application areas.

Let us turn to the theoretical foundation of the MAP queries. The following theorem combines the previous results and formulates the log-linear $\mathcal{EL}^{++}$ MAP query as a maximization problem subject to a set of logical constraints.

Theorem 7. *Let $\mathcal{C} = (\mathcal{C}^{\mathsf{D}}, \mathcal{C}^{\mathsf{U}})$ be a $\mathcal{EL}^{++}$ CBox, let N_{U} be the set of concept and role names used in $\mathsf{norm}_{\mathsf{LL}}(\mathcal{C})$, and let $\mathcal{H}$ be the Herbrand base of $\mathcal{F}$ with respect to N_{U}. Moreover, let $\mathcal{K}$ be the set of ground formulas constructed from $\mathcal{C}^{\mathsf{D}}$ and let $\mathcal{G}$ be the set of weighted ground formulas constructed from $\mathcal{C}^{\mathsf{U}}$. Then, with*

$$\hat{H} := \underset{\{H \subseteq \mathcal{H} \colon \models_H (\mathcal{K} \cup \mathcal{F})\}}{\operatorname{argmax}} \sum_{\{(G, w_G) \in \mathcal{G} \colon \models_H G\}} w_G \tag{8.1}$$

we have that $\varphi^{-1}(\hat{H})$ is a most probable coherent CBox over N_{U} that entails $\mathcal{C}^{\mathsf{D}}$.

Proof. We show that $\varphi^{-1}(\hat{H})$ is a (a) *most probable* (b) *coherent* CBox over N_{U} that (c) entails $\mathcal{C}^{\mathsf{D}}$.

(a) The MAP query computes a most probable world. In Section 2.1.1 and Section 2.1.3 we defined the MAP query in log-linear models as followed. Let $\mathbf{x}$ be a truth assignment to variables $\mathcal{X}$. Let $f_i(x)$ be binary features associated with a weight w_i. Then, the MAP query computes the most probable world with

$$\mathbf{x} = \underset{\mathbf{x}}{\operatorname{argmax}} \ \frac{1}{Z} \exp\left(\sum_i w_i f_i(\mathbf{x})\right).$$

Since $\frac{1}{Z}$ is constant and the exponential function is a monotonous increasing function, we obtain

$$\mathbf{x} = \frac{1}{Z} \exp\left(\underset{\mathbf{x}}{\operatorname{argmax}} \ \sum_i w_i f_i(\mathbf{x})\right) = \underset{\mathbf{x}}{\operatorname{argmax}} \ \sum_i w_i f_i(\mathbf{x}).$$

We now associate each binary weight feature pair $f_i(x)$ and w_i with G and w_g such that $f_i(x) = \begin{cases} 1 & if\ (G, w_G) \in \mathcal{G} : \vDash_H G \\ 0 & otherwise \end{cases}$. Concequently, $\hat{H}$ is a most probable world and $\varphi^{-1}(\hat{H})$ a most probable CBox.

(b) We know from Lemma 5 that for any $H' \subseteq \mathcal{H}$ we have that if H' is a Herbrand model of $\mathcal{F}$ then $\varphi^{-1}(H')$ is a classified and coherent CBox. From $\hat{H}$ we know that $\hat{H} \subseteq \mathcal{H}$. Furthermore, $\hat{H}$ entails $\mathcal{K} \cup \mathcal{F}$. From this follows that $\hat{H} \vDash_H \mathcal{F}$ and thus $\hat{H}$ is a Herbrand model of $\mathcal{F}$. Consequently, $\hat{H}$ is classified and coherent.

(c) Since $\hat{H} \vDash_{\hat{H}} (\mathcal{K} \cup \mathcal{F})$ we can conclude that $\hat{H} \vDash_{\hat{H}} (\mathcal{K})$. Since $\mathcal{C}^{\mathsf{D}} = \varphi(\mathcal{K})$ we can conclude that $\varphi^{-1}(\hat{H})$ entails $\mathcal{C}^{\mathsf{D}}$.

□

Theorem 7 illustrates how we can compute the maximum a-posteriori (MAP) query in log-linear description logics to retrieve the most probable coherent CBox. Intuitively, the theorem uses the general definition of a MAP query from Section 2.1.3 and constructs a specific maximum a-posteriori query. This query returns a CBox which maximizes the weights of the uncertain CBox $\mathcal{C}^{\mathsf{U}}$, entails the deterministic CBox $\mathcal{C}^{\mathsf{D}}$, and is coherent. Please note that the MAP query derives a most probable CBox *and* classifies it at the same time.

In order to be able to perform this computation, we have to (a) transform the log-linear CBox to logical formulas, (b) compute the MAP query based on these formulas, and, (c) transform the result back into description logic axioms. For (a) the mapping from description logics to logical formulas from the last section is reused. The logical representation depict the set $\mathcal{K}$, which contains the literals constructed from the deterministic CBox $\mathcal{C}^{\mathsf{D}}$, and the set $\mathcal{G}$, which contains weighted conjunctions derived from the uncertain CBox $\mathcal{C}^{\mathsf{U}}$. In (b) we search the Herbrand base $\hat{H}$ which maximizes the sum of the weights w_G of each $G \in \mathcal{G}$. Every of such G has to be entailed in $\hat{H}$. Please note, that we use a capital G here to emphasize

that G contains conjunctions of literals. Furthermore, this Herbrand base has to entail all literals in $\mathcal{G}$ and the ground formulas constructed out of the hard rule set $\mathcal{F}$. This hard rule set $\mathcal{F}$ consist of the formulas listed in Table 8.5. After we computed the Herbrand base $\hat{H}$ we can, finally, transform it back to the description logic view (step (c)) by the function $\varphi^{-1}(\hat{H})$ and result in the most probable coherent CBox.

8.3.5 Transformation to a Markov Logic Network

The mapping from log-linear DL to logic and back has been exhaustively defined in the last sections. Thus, we now require to develop a solving technique for the concrete MAP query given in Theorem 7. For this, we translate the theorem into a concrete Markov logic network (MLN) and can then reuse the efficient ways of solving MAP queries by translating it into integer linear programs presented in Part I. In theory, the previous sections already build a Markov network. However, the following algorithm wraps up the theoretical findings, illustrates the interconnection between previous components.

Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be sets of concept and role names, respectively. Let $\mathcal{K}$ be the set of ground formulas constructed from $\mathcal{C}^{\mathsf{D}}$ and let $\mathcal{G}$ be the set of weighted ground formulas constructed from $\mathcal{C}^{\mathsf{U}}$. Then, we construct the following MLN network.

1. We encode every literal $\ell \in \mathcal{K}$ as evidence.

2. We introduce the predicates sub and $rsup$ as hidden predicates and the predicates int, $rsub$, $psub$, and $pcom$ as observed predicates. This means, that for the latter predicates all literals which are not encoded in the evidence are implicitly set to $false$.

3. We add each formula $f \in \mathcal{F}$ as hard clause to the MLN.

4. We construct the two types $\mathcal{T}_C$ and $\mathcal{T}_R$, where $\mathcal{T}_C$ contains constants representing all concept names in $\mathsf{N_C}$ and $\mathcal{T}_R$ contains all role name constants from $\mathsf{N_R}$, respectively. We then use those types to restrict the possible constants for variables of the formulas of step (3).

5. We add the weighted ground formulas $\langle G, w_G \rangle \in \mathcal{G}$ to the MLN.

For details about the components of a MLN we refer the reader to Section 2.3. Especially for the definition of types as well as hidden and observed predicates we recommend to read Section 2.3.1.

After this transformation, we can compute the MAP query with the techniques presented in Part I. For step (5) we require the ability to compute the MAP queries for weighted conjunctions. To that end, we require to extend Markov logic such that it can handle weighted conjunctions. This extension will be explained in Section 8.4. Example 27 illustrates the MLN construction and provides the MAP query result.

Example 27. *Let us recapture the log-linear ontology $\mathcal{O}_2^{LL}$ from Example 26. The concept and role names are $\mathsf{N_C} = \{Animal, Tiger, Plant\}$ and $\mathsf{N_R} = \{eats\}$. For easier replicability, we print the two sets $\mathcal{K}$ and $\mathcal{G}$ again:*

$\mathcal{K}$	$int(Animal, Plant, \bot)$	*(1)*
$\mathcal{G}$	$\langle sub(Tiger, Animal) \wedge \ldots \wedge sub(X_{eats,Plant}, Plant), 1.1\rangle$	*(2)*
	$\langle sub(X_{eats,A_1}, Animal) \wedge sub(X_{eats,Plant}, Animal), 0.3\rangle$	*(3)*

For brevity we omitted some translations of axiom (2) and replaced them with dots. We refer to Example 26 for a full list of axioms.

We now transform the log-linear ontology $\mathcal{O}_2^{LL}$ into a MLN performing the following steps:

1. *We encode $\{int(Animal, Plant, \bot)\} \in \mathcal{K}$ as evidence.*

2. *We set the hidden and observed predicates as described.*

3. *We add every formula in $\mathcal{F}$ from Table 8.5 as hard clauses:*
   F_1 $\langle sub(c, c), \infty\rangle$
   F_2 $\langle sub(c, \top), \infty\rangle$
   F_3 $\langle \neg sub(c_1, c_2) \vee \neg sub(c_2, c_3) \vee sub(c_1, c_3), \infty\rangle$
   F_4 $\forall c_1, c_2, c_3, c_4 : sub(c_1, c_2) \wedge sub(c_1, c_3) \wedge int(c_2, c_3, c_4) \Rightarrow sub(c_1, c_4)$
   ...
   F_9 $\langle \neg sub(c, \bot), \infty\rangle$

4. *After construction, the two types contain the constants $\mathsf{T_C} = \{Animal, Plant, Tiger\}$ and $\mathsf{T_R} = \{eats\}$. Each variable c, c_i is now associated with type $\mathcal{T}_C$ and each variable r_i with type $\mathcal{T}_R$ (with $1 \leq i \leq 4$) in each Formula.*

5. *We add the following weighted conjunctions to the MLN:*
   $\langle sub(Tiger, Animal) \wedge \ldots \wedge sub(X_{eats,Plant}, Plant), 1.1\rangle$
   $\langle sub(X_{eats,A_1}, Animal) \wedge sub(X_{eats,Plant}, Animal), 0.3\rangle$

If we compute the MAP query of the MLN, we reach the objective 1.1. This means that the most-probable coherent ontology entails axioms (1) and (2) but not axiom (3). Axiom (3) is in conflict with axiom (2) since rule F_4 derives

$$sub(X_{eats,Plant}, Plant) \wedge sub(X_{eats,Plant}, Animal) \wedge int(Animal, Plant, \bot) \Rightarrow sub(X_{eats,Plant}, \bot).$$

Since (2) has a higher weight than (3) the MAP state contains axiom (2) and not axiom (3).

8.4 Extension of Markov Logic to Formulas in Conjunctive Normal Form

The correct translation of log-linear description logic to Markov logic networks requires the ability to assign weights to conjunctions of literals. However, traditional Markov logic assigns weights only to disjunctions of literals (called clauses). If the conjunctive normal form of a weighted first-order formula contains more than one clause, then Richardson and Domingos [RD06] divide the associated weight through the number of clauses and split the clauses in individual formulas. Example 28 illustrates this procedure and shows the consequences.

Example 28. *We have the following weighted first-order formula taken from [RD06]:*

$$\langle Friends(x,y) \Rightarrow Smokes(x) \Leftrightarrow Smokes(y), 2.2\rangle \quad (1)$$

The conjunctive normal form of this first-order formula is:

$$\langle(\neg Friends(x,y) \vee Smokes(x) \vee \neg Smokes(y)) \wedge \quad (2)$$
$$(\neg Friends(x,y) \vee \neg Smokes(x) \vee Smokes(y)), 2.2\rangle$$

The approximation from [RD06] splits this in the following two clauses:

$$\langle \neg Friends(x,y) \vee Smokes(x) \vee \neg Smokes(y), 1.1\rangle \quad (3)$$
$$\langle \neg Friends(x,y) \vee \neg Smokes(x) \vee Smokes(y), 1.1\rangle \quad (4)$$

Let us now assume that we have a possible world

$$\mathbf{x} = \{Friends(A,B), Smokes(A), \neg Smokes(B)\}.$$

If we use the original translation (2), we receive the probability

$$P(\mathbf{x}) = \frac{1}{Z} \exp(2.2 \cdot 0) = \frac{1}{Z}.$$

However, if we use the approximation of clause (3) and (4), we recieve for the same world $\mathbf{x}$ *the probability*

$$P(\mathbf{x}) = \frac{1}{Z} \exp(1.1 \cdot 1 + 1.1 \cdot 0) = \frac{1}{Z} \exp 1.1.$$

Thus, this approximation leads to different probabilities of the same possible world.

In Example 28 we have gained an intuition that this approximation often leads to different semantic representations. In this Section, we extend the semantics of Markov logic such that they can deal with arbitrary weighted CNF formulas. Furthermore, we extend our optimized ILP translation (see Section 3.2) and our novel CPA approach (see Section 3.4) from Part I to work with conjunctions of literals.

In particular, we first recall an approach, which creates new predicates for each clause in a conjunctive normal form and connects this predicate with a hard rule to the results of this clause [FdBR$^+$13, Jan04]. Replacing the clauses by its representing predicates, results in a conjunction of literals. Second, we show how conjunctions of literals can be translated to ILP constraints (in Section 8.4.2) and integrated into the CPA and CPI aggregation (in Section 8.4.3).

Please note that the generalization of integer translation to weighted CNFs is a side-product of this thesis. Extending the CPA algorithm with conjunction is novel and has not been published and experimentally verified yet. The idea of splitting the weighted CNF formulas into separate conjunctions of disjunctions have been introduced in [FdBR+13, Jan04].

8.4.1 Reduction to Conjunctions

Our goal is to semantically interpret a weighted CNF formula f correctly. This formula f has the form

$$\langle f, w_f \rangle = \langle c_1 \wedge \ldots \wedge c_n, w_f \rangle$$

where each c_i is a clause of the form $c_i = \ell_{i,1} \vee \ldots \vee \ell_{i,m}$.

To that end, we first simplify the formula to be a conjunction of literals by introducing a new predicate p_{c_i} for each clause c_i. This reduction is inspired by existing methods [FdBR+13, Jan04]. We refer the reader to this literature for formal justifications.

Let $\mathcal{X}(c_i)$ be the set of variables of clause c_i. Then, we construct the novel literal $p_{c_i}(x_1, \ldots, x_n)$ out of the variables $x_1, \ldots, x_n \in \mathcal{X}(c_i)$. Consequently, the arity of predicate p_{c_i} equals the size of $\mathcal{X}(c_i)$. Trivially, if the clause c_i consists of only one literal, we do not have to introduce a new predicate. Additionally, we have to link the new literal $p_{c_i}(x_1, \ldots, x_n)$ with the clause c_i, which is translated to the logical statement

$$(\ell_{i,1} \vee \ldots \vee \ell_{i,m}) \Leftrightarrow p_{c_i}(x_1, \ldots, x_n).$$

Similar to Section 3.2, we distinguish between positive and negative weights. This prevents us from having to model both directions. If $w_f > 0$ we only have to encode the direction $\Rightarrow$ which leads to multiple hard clauses

$$\langle \neg\ell_{i,1} \vee p_{c_i}(x_1, \ldots, x_n), \infty \rangle, \ldots, \langle \neg\ell_{i,m} \vee p_{c_i}(x_1, \ldots, x_n), \infty \rangle.$$

This hard clauses model the implication: if c_i, then $p_{c_i}(x_1, \ldots, x_n)$. This means that $p_{c_i}(x_1, \ldots, x_n)$ can not be *true* if c_i is *false*. Since the weight is positive, $p_{c_i}(x_1, \ldots, x_n)$ always *tries* to become *true* to maximize the weights, and thus have to be forced to *false* if c_i is false.

If $w_f < 0$ we have to encode the $\Leftarrow$ direction which leads to the hard clause

$$\langle \ell_{i,1} \vee \ldots \vee \ell_{i,m} \vee \neg p_{c_i}(x_1, \ldots, x_n), \infty \rangle.$$

Respectively, the clause now models the implication: if $p_{c_i}(x_1, \ldots, x_n)$, then c_i. Since the weight is negative, $p_{c_i}(x_1, \ldots, x_n)$ *tries* to become *false* if possible to avoid the realization of the negative weight. However, our constraint forces $p_{c_i}(x_1, \ldots, x_n)$ to be *true* whenever c_i is *true*.

Since these are all *normal* hard clauses, they can be translated to the ILP like any other hard clause. We refer the reader to Section 3.2 for details.

Finally, we replace each clause c_i with its literal $p_{c_i}(x_1, \ldots, x_n)$ which result in a conjunction of literals

$$\langle \bigwedge_i p_{c_i}(x_1, \ldots, x_n), w_f \rangle.$$

Example 29 illustrates the transformation process. This transformation can be performed in polynomial time [Jan04]. Although in some cases many hard formulas have to be created, the computation of hard formulas is usually very efficient when utilizing the CPA and the CPI algorithm. We are now in the position that we only have to come up with new ILP translations for conjunctions to cope with general weighted CNF formulas.

Example 29. *In this example we reuse the first-order formula f in CNF form from Example 28*

$$\langle (\neg Friends(x,y) \vee Smokes(x) \vee \neg Smokes(y)) \wedge \quad (2)$$
$$(\neg Friends(x,y) \vee \neg Smokes(x) \vee Smokes(y)), 2.2 \rangle$$

to illustrate the reduction to conjunctions. We introduce two new predicates p_1 and p_2 and add the following two hard clauses to the MLN:

$$\langle Friends(x,y) \vee p_1(x,y), \infty \rangle, \langle \neg Smokes(x) \vee p_1(x,y), \infty \rangle,$$
$$\langle Smokes(y) \vee p_1(x,y), \infty \rangle$$
$$\langle Friends(x,y) \vee p_2(x,y), \infty \rangle, \langle Smokes(x) \vee p_2(x,y), \infty \rangle,$$
$$\langle \neg Smokes(y) \vee p_2(x,y), \infty \rangle$$

Finally, we transformed the soft CNF formula f to hard rules and the following soft conjunction rule

$$\langle p_1(x,y) \wedge p_2(x,y), 2.2 \rangle.$$

8.4.2 Integer Linear Program Translation of Conjunctions

The previous section provides us a conjunction which needs to be translated in ILP constraints. To that end, we first have to ground the conjunction. This works similarly to the grounding of clauses, which is explained in Section 2.3.2, except that the rules for omitting literals or clauses due to evidence change. Since those changes are straight forward we do not describe them here.

From now on we assume that we have given a set $\mathcal{H}$ containing all conjunctions of grounded literals h with

$$h = \ell_i \wedge \ldots \wedge \ell_n.$$

associated with a weight w_h which can be either infinitive or a real number. We use the letter h for conjunctions of grounded literals and the letter g for clauses. From now on, we shorten the term conjunctions of grounded literals by just saying grounded conjunctions. The literals ℓ_i $(1 \leq i \leq n)$ can be negated or unnegated.

For translating h into ILP constraints, we distinguish between the cases where weight w_h is infinitive (hard formula), positive, and negative. This is done analogous to Section 3.2.

One binary ILP variable x_ℓ is associated with each ground atom ℓ occurring in the set of all grounded conjunctions and all ground clauses. For a ground conjunction h let $L^+(h)$ be the set of ground atoms occurring unnegated in h and $L^-(h)$ be the set of ground atoms occurring negated in h.

For every ground conjunction $h \in \mathcal{H}$ with weight $w > 0,\ w \in \mathbb{R}$, we add a novel binary variable z_h, the term $w_h \cdot z_h$ to the objective, and the following constraint to the ILP:

$$\sum_{\ell \in L^+(h)} x_\ell + \sum_{\ell \in L^-(h)} (1 - x_\ell) \geq (|L^+(h)| + |L^-(h)|) z_h.$$

For every h with weight $w_h < 0,\ w \in \mathbb{R}$, we add a novel binary variable z_h, the term $w_h \cdot z_h$ to the objective, and the following constraint to the ILP:

$$\sum_{\ell \in L^+(h)} x_\ell + \sum_{\ell \in L^-(h)} (1 - x_\ell) \leq (|L^+(h)| + |L^-(h)| - 1) + z_h.$$

For every h with weight $w_h = \infty$, we add the following constraint to the ILP:

$$\sum_{\ell \in L^+(h)} x_\ell + \sum_{\ell \in L^-(h)} (1 - x_\ell) \geq |L^+(h)| + |L^-(h)|.$$

The idea behind the translation of conjunctions to ILP constraints is analogous to the translation of disjunctions of literals (also called clauses) in Section 3.1 and Section 3.2. Thus, we refer the reader to these sections for comments on the ILP constraints.

8.4.3 Extending the Cutting Plane Aggregation Approach With Conjunctions

Conjunctions are integrate-able into the cutting plane inference (CPI) and cutting plane aggregation (CPA) approaches. The algorithm for CPI does not change when including conjunctions. However, violated constraints of conjunctions are of course detected differently than for disjunctions. For details about violated constraints for conjunctions we forward the reader to Section 4.2.

Weighted conjunctions can also be aggregated with our new CPA approach. The aggregation approach is similar to the aggregation of clauses. Thus, Definition 1 also holds for conjunctions. Consequently, we end up with a set $H \subset \mathcal{H}$ of n weighted conjunctions of grounded literals, which are aggregated with respect to c. In our case c is a conjunction of grounded literals. As before, we name the non-aggregated literal ℓ_i.

However, we obtain differences in the semantic. Thus, it requires to set up a new version of Lemma 2 which is as follows:

Lemma 8. *Let $H \subseteq \mathcal{H}$ be a set of conjunctions of ground literals (called ground conjunction) with weight w and let c be a ground conjunction. Moreover, let us assume that H can be aggregated with respect to c, that is, that each $h \in H$ can be written as $\ell_i \wedge c$. The aggregated feature f^H for the aggregated conjunction H with weight w maps each interpretation I to an integer value as follows*

$$f^H(I) = \left\{ \begin{array}{cc} |\{\ell_i \wedge c \in H \mid I \models \ell_i\}| & if \;\; I \models c \\ 0 & otherwise \end{array} \right\}.$$

Proof. The proof is analogous to the proof of Lemma 2. We first define the individual feature $f^h(I)$ as

$$f^h(I) = \left\{ \begin{array}{ll} 1 & if\ I \models h \\ 0 & otherwise \end{array} \right. .$$

The features can be aggregated to

$$f^H(I) = \sum_{h \in H} f^h(I)$$

since all $h \in H$ share the same weight w. Again, we distinguish two cases:

$I \not\models c$: If $I \not\models c$ we can conclude that no $h \in H$ is satisfied since each $h \equiv \ell_i \wedge c \equiv \ell_i \wedge false \equiv false$. In this case, each individual feature function returns $f^h(I) = 0$ and thus the aggregated feature function must return $f^H(I) = \sum_{h \in H} f^h(I) = 0$.

$I \models c$: Since each $h \in H$ is equivalent to $\ell_i \wedge c \equiv \ell_i \wedge true \equiv \ell_i$, it is a requirement for h being satisfied that c is satisfied . Under the condition that $I \models c$, the individual feature function $f^h(I)$ of each clause h returns

$$f^h(I) = \left\{ \begin{array}{ll} 1 & if\ I \models \ell_i \\ 0 & otherwise \end{array} \right. .$$

Finally, we rewrite the aggregated feature function as

$$f^H(I) = \sum_{h \in H} f^h(I) = |\{\ell_i \wedge c \in H \mid I \models \ell_i\}|.$$

Please note that this lemma is also valid for sets of ground conjunctions H containing only one h. In this case, the individual feature $f^h(I)$ equals the aggregated feature $f^H(I)$. □

The feature resulting from the aggregation, therefore, counts the number of literals ℓ_i that are satisfied whenever the ground conjunction c is satisfied. If the conjunction c is not satisfied, every conjunction evaluates to false. Thus, the feature function returns zero in this case.

Consequently, the ILP translation is different for aggregated conjunctions H than for clauses.

For any ground conjunction c, let $L^+(c)$ ($L^-(c)$) be the set of ground atoms occurring unnegated (negated) in c. Let $H \subseteq \mathcal{H}$ be a set of n ground conjunctions that can be aggregated with respect to c, that is, for each $h \in H$ we have that $h = x_i \wedge c$ or $h = \neg x_i \wedge c$ for a ground atom x_i and a fixed clause c. Additionally, we introduce $n = |H|$. We now introduce a new *integer* variable z_H and add the term $w_h \cdot z_H$ to the objective.

In case the weight is *positive* ($w > 0$) we add the following constraints to the ILP:

$$\sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1 - x_i) \geq z_H \tag{8.2}$$

$$n x_\ell \geq z_H \text{ for every } \ell \in L^+(c) \tag{8.3}$$

and

$$n(1 - x_\ell) \geq z_H \text{ for every } \ell \in L^-(c) \tag{8.4}$$

The only syntactical difference to the case of clauses with negative weights is the use of $\geq$ instead of $\leq$. The constraint formulation follows the intuition that in conjunctions every non-negated literal has to be satisfied, and every negated literal must not be satisfied. Thus, our integer variable z_H is only allowed to be greater than zero, if every (negated) literal in the aggregated conjunction c is (not) satisfied. This is ensured by Equation 8.3 and Equation 8.4, respectively. If c is satisfied, Equation 8.2 ensures that z_H is at most the number of (non-)satisfied (negated) literals.

In case the weight is *negative* ($w < 0$) we add the constraint

$$\sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) - n \cdot m + \sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1 - x_i) \leq z_H \tag{8.5}$$

where $m = |L^+(c)| + |L^-(c)|$ is the number of literals in c. The idea behind this restriction is a little bit tricky. Since we are dealing with negative weights, the integer variable z_H intuitively wants to be as small as possible. This means, we have to ensure that z_H is at least as high as the output of function $f^H(I)$.

In case all $\ell \in L^+(c)$ ($\neg\ell \in L^-(c)$) are satisfied, the constraint simplifies to

$$n \cdot m - n \cdot m + \sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1 - x_i).$$

Thus, we basically count the true $\ell \in L^+(c)$ ($\neg\ell \in L^-(c)$) literals.

Next, we examine the case where not all $\ell \in L^+(c)$ ($\ell \in L^-(c)$) are true (false). Correspondingly, the left side of our constraint has a result smaller or equal than 0. The reason for this is that

$$\sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) - n \cdot m \leq -n,$$

since at least one of the m literals $\ell \in L^+(c)$ ($\neg\ell \in L^-(c)$) is not satisfied.

For hard constraints with *infinite* weights ($w = \infty$), we add the constraint:

$$\sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) + \sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1 - x_i) \geq n \cdot (m + 1) \quad (8.6)$$

where $m = |L^+(c)| + |L^-(c)|$ is the number of literals in c. This constraint models the requirement that every literal $\ell \in L^+(c)$ ($\neg\ell \in L^-(c)$) must be satisfied as well as every literal x_i of form $(x_i \vee c) \in H$ ($\neg x_i$ of form $(\neg x_i \vee c) \in H$) must be satisfied.

For our extended Markov logic semantic, we require an updated version of Theorem 3 which is as followed.

Theorem 9. *Let $\mathcal{M}$ be a Markov logic network allowing weighted formulas in conjunctive normal form. Furthermore, let ILP($\mathcal{M}$) be the ILP formulation with aggregated cutting planes. Each solution of ILP($\mathcal{M}$) corresponds one-to-one to a maximum a-posteriori state of the Markov logic network $\mathcal{M}$.*

Proof. The conversion of weighted CNF to hard clauses and weighted conjunctions have been shown by [Jan04]. Theorem 3 showed the correctness for clauses. In order to proof the theorem above, we have to show that our ILP translation of the aggregation of conjunctions correctly translates the aggregated features from Lemma 8.

Let $H \subseteq \mathcal{H}$ be a set of aggregated ground conjunctions with weight w. Moreover, let us assume that H can be aggregated with respect to conjunction c, that is, that each $h \in H$ can be written as $\ell_i \vee c$. Let I be any interpretation. For each aggregated feature $f^H(I)$, we introduced an integer ILP variable z_G and added $w \cdot z_G$ to the objective. We now have to show that $z_H = f^H(I)$.

In case of $w > 0$, the ILP constraints in Formula 8.2, Formula 8.3, and Formula 8.4 ensure that $z_H = f^H(I)$.

- If $I \models c$, then all literals $\ell \in L^+(c)$ ($\neg\ell \in\in L^-(c)$) are satisfied and Formula 8.3 and Formula 8.4 restrict z_H to $z_H \leq |H|$. However, Formula 8.2 further restricts z_H to $z_H \leq |\{\ell_i \wedge c \in H \mid I \models \ell_i\}|$. Due to $w > 0$ the ILP tries to maximize z_H. Thus, we can conclude $z_H = f^H(I) = |\{\ell_i \wedge c \in H \mid I \models \ell_i\}|$.

- If $I \not\models c$, at least one literal $\ell \in L^+(c)$ ($\neg\ell \in\in L^-(c)$) is satisfied. Thus, at least one instance of Formula 8.3 or Formula 8.4 ensure that $z_H \leq 0$. Thus, we obtain $z_H = f^H(I) = 0$.

In case of $w < 0$, the ILP constraint in Formula 8.5 ensures that $z_G = f^G(I)$.

- If $I \models c$, then then all literals $\ell \in L^+(c)$ ($\neg\ell \in\in L^-(c)$) are satisfied and Formula 8.5 simplifies to

$$n \cdot m - n \cdot m + \sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1 - x_i) \leq z_H.$$

The remaining part ensures that $z_H \geq |\{\ell_i \wedge c \in H \mid I \models \ell_i\}|$. Since $w < 0$ the ILP tries to minimize z_H and we conclude $z_H = f^H(I) = |\{\ell_i \wedge c \in H \mid I \models \ell_i\}|$.

- If $I \not\models c$, at least one literal $\ell \in L^+(c)$ ($\neg\ell \in\in L^-(c)$) is not satisfied. Thus, we can infer from Formula 8.5 that

$$\sum_{\ell \in L^+(c)} nx_\ell + \sum_{\ell \in L^-(c)} n(1-x_\ell) - n \cdot m + \sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1-x_i) \leq$$

$$(n-1) \cdot m - n \cdot m + \sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1-x_i) =$$

$$-n + \sum_{(x_i \vee c) \in H} x_i + \sum_{(\neg x_i \vee c) \in H} (1-x_i) \leq$$

$$-n + n \leq 0 \leq z_H.$$

Since $w < 0$ the ILP minimizes z_H and thus we get $z_H = f^H(I) = 0$.

□

With this extension of our cutting plane aggregation methodology, we are now able to aggregate weighted conjunctions. This allows us to efficiently solve the Markov logic networks which are constructed for computing the most probable coherent world in log-linear description logics.

Chapter 9

Related Work

In the following, we describe several description logics which can deal with some kind of uncertainty. According to Lucasiewicz and Straccia [LS08] they can be categorized into probabilistic, possibilistic, and fuzzy description logics. We selected the approaches according to their degree of importance in the community and if a usable implementation exists. To the best of our knowledge, Section 9.4 contains all existing systems which combine any description logic with some kind of uncertainty 9.4.

Throughout the description of existing uncertain description logics our aim is to compare them with log-linear description logic. In particular, we want to answer the question if any other uncertain description logic is able to compute a most-probable coherent ontology.

The broad structure in probabilistic, possibilistic, and fuzzy description logics as well as some explanations are inspired by [LS08]. However, most content result from an extensive literature research of various different sources. Despite [LS08], the main sources within Section 9.1 have been: [Luk08] for Section 9.1.1, [FdBT$^+$11, RKT07] for Section 9.1.2, and [dCL06, LdC12, Las08] for Section 9.1.3. For Section 9.2 the main sources where [DLP94, QJPD10] and Section 9.3 is mainly summarized from [LS08]. Section 9.4 is based on several publications describing the respective system.

9.1 Probabilistic Description Logics

Before we describe the existing approaches in probabilistic description logics we briefly recall the main properties of probabilistic logic. For illustration, we present the classic probability logic described in [Nil86] and summarized in [LS08]. Beyond this, there have been published a large variety of formal languages based on probabilistic logic. For more probabilistic languages we refer to the surveys [Háj01, Hai96].

Let ϕ and ψ be basic events. Then, other events can be recursively constructed by the connectives $\neg\phi$, $\phi \vee \psi$, and $\phi \wedge \psi$. A probabilistic formula assigns a real

number $p \in [0, 1]$ to an event. Intuitively, e.g. $rain\_tomorrow \geq 0.7$ means that it will rain tomorrow with a probability of 0.7. Finally, a probabilistic knowledge base is a set of probabilistic formulas. We skip the detailed definitions of the semantics here. For details about the semantics, the reader is referred to [Nil86]. However, we want to briefly illustrate the computation of probabilities in order to be able to point out the difference to possibilistic logic later in Section 9.2. A probability interpretation Pr is a probability function which maps a concrete world to a probability. Hereby, the following relationships hold:

$$Pr(\phi \wedge \psi) = Pr(\phi) + Pr(\psi) - Pr(\phi \vee \psi);$$
$$Pr(\phi \vee \psi) = Pr(\phi) + Pr(\psi) - Pr(\phi \wedge \psi);$$
$$Pr(\neg\psi) = 1 - Pr(\phi).$$

In the following, we concentrate on three approaches which represent the three main variants of combining probabilistic models with description logics. First, Section 9.1.1 presents $\mathcal{P}$ – $\mathcal{SROIQ}$ [Luk08] which defines conditional probabilities about concepts and statements expressing uncertain knowledge about objects. Then, Section 9.1.3 describes PR-OWL [Cos05, dCL06] which is based on Bayesian networks. Finally, we introduce a semantic based on probabilistic prolog [RKT07, FdBT$^+$11] in Section 9.1.2.

Not included in the following sections is the the probabilistic description logic defined in [Luk07]. They combine description logic programs under answer set programming [EIL$^+$08] and well-founded semantics with independent choice logic [Poo08]. For query processing all answer sets have to be computed out of which an integer linear program is constructed.

In delimitation to log-linear description logics, non of these approaches build their semantics on Markov logic networks and compute the most probable coherent ontology as it was introduced in Chapter 8.2.

9.1.1 $\mathcal{P}$ – $\mathcal{SROIQ}$

To the best of our knowledge, the probabilistic description logic $\mathcal{P}$ – $\mathcal{SROIQ}$ [Luk08] is the most expressive description logics that combine conditional probabilities about concepts and statements expressing uncertain knowledge about instances. The first provides statistical information about a large number of instances (also called terminological probabilistic knowledge), while the latter one defines a particular degree of belief for one specific instance (also called assertional probabilistic knowledge). Each of the probability statements has two probabilities attached where the first one represents the probability to which the statement is at least true and the second one is the probability to which the statement is at most true. The terminological probabilistic statement $(HasFourWheels|Car)[0.9, 1]$ for instance means that cars have four wheels with a probability of at least 0.9. A $\mathcal{P}$ – $\mathcal{SROIQ}$ terminological box consists of a classical knowledge base T and a finite set of terminological probabilistic statements P. The assertional probabilistic statements P_o are created relative to an individual o. The statement 'John's car is a

sports car with probability of at least 0.8' is expressed by the conditional constraint $(SportsCar|\top)[0.8, 1] \in P_{John's car}$. Thus, each individual in a $\mathcal{P}-\mathcal{SROIQ}$ assertional box can have a a finite set of assertional probability statements assigned.

In $\mathcal{P}-\mathcal{SROIQ}$ there exist three main reasoning problems [Luk08, LS08]:

Probabilistic termionological box consistency. Given a probabilistic terminological knowledge base PT with a classical knowledge base T and a finite set of terminological probabilistic statements P, decide if PT is consistent. Informally, PT is consistent if any inconsistencies can be naturally resolved by preferring more specific pieces of knowledge to less specific ones.

Probabilistic knowledge base consistency. The probabilistic knowledge base is consistent if the terminological knowledge base PT is consistent and if $T \cup P_o$ is satisfiable for every probabilistic individual o. Thus, the consistency of the terminological knowledge base PT and the and the satisfiablility of the probabilistic individuals P_o is done separately.

Tight lexicographic entailment. Intuitively, tight lexicographic entailment infers as much new knowledge as possible so that we prefer more specific pieces of knowledge to less specific ones in case of local inconsistencies. For terminological tight lexicographic entailment, we search for the maximal number of terminological probabilistic statements P that are entailed in PT. The same idea is applied to the assertional box.

Example 30 (inspired by [LS08]) illustrates these three reasoning types.

Example 30. *Let us assume that* $T = \{SportsCar \sqsubseteq Car\}$ *and*

$$\begin{aligned} P = \{&(HasFourWheels|Car)[0.9, 1], \\ &(\exists HasSeats.\{four\}|Car)[0.9, 1], \\ &(\neg\exists HasSeats.\{four\}|SportsCar)[0.7, 1]\}. \end{aligned}$$

When we now reason about $SportsCar$ *the inconsistency can be resolved by preferring the more specific statement* $(\neg\exists HasSeats.\{four\}|SportsCar)[0.7, 1]$ *over the less specific statement* $(\exists HasSeats.\{four\}|Car)[0.9, 1]$. *Thus, our probabilistic termionological box is consistent.*

Let us now define an assertional box consisting only of one individual John's car and containing the conditional probability statement

$$(SportsCar|\top)[0.8, 1] \in P_{John's\ car}.$$

Then, our whole probabilistic knowledge base is again consistent since the probabilistic terminological box is consistent and $T \cup P_{John's\ car}$ *is consistent. In terminological tight lexical entailment the new axiom*

$$(HasFourWheels|SportsCar)[0.9, 1]$$

can be inferred. New tight lexical consequences for the individual John's car are the axioms $(\neg\exists HasSeats.\{four\}|\top)[0.7, 1]$ *and* $(HasFourWheels|\top)[0.72, 1]$.

Consistency checking of $\mathcal{P}$ – $\mathcal{SROIQ}$ can be reduced to the problem whether a classical knowledge base is decidable and whether a system of linear constraints is solvable. In particular, a sequence of problems have to be solved, where the variables of the linear constraints are computed by deciding classical knowledge base satisfiability problems. Tight lexical entailment can solved with a similar technique. However, the complexity is much higher. We refer the interested reader to [Luk08] for details and precise complexity results. The system PRONTO [Kli11] implements these inference algorithms including some optimizations (see Section 9.4).

A predecessor of the presented probabilistic description logic $\mathcal{P}$ – $\mathcal{SROIQ}$ is the less expressive probabilistic description logic $\mathcal{P}$ – $\mathcal{SHOQ}$ [GL02]. Furthermore, the probabilistic logics in [Hei94, Jae94] are related. However, the probabilistic variant of $\mathcal{ALC}$ [Hei94] does not allow inference about assertions while the logic introduced by Jaeger et al. [Jae94] only allows concept assertions but mention the possibility to extend their approach to role assertions. The presented probabilistic description logic $\mathcal{P}$ – $\mathcal{SROIQ}$ allows both, role and concept assertions. Jaeger et al. [Jae94] presents an inference technique where they combine terminological and assertional probabilistic knowledge into one probability space and apply cross entropy minimization. Cross entity minimization is a measure of information dissimilarity of two probability measures [SJ81].

Finally, we conclude that the semantic and the query tasks of $\mathcal{P}$ – $\mathcal{SROIQ}$ are not comparable with the semantics of log-linear description logic and with the query of finding the most probable coherent ontology presented in Chapter 8.

9.1.2 DISPONTE

The semantic DISPONTE [BLRA11] (Distribution Semantics for Probabilistic Ontologies) is closely related to the probabilistic logic $\mathcal{P}$ – $\mathcal{SROIQ}$ (see previous section). According to [LS08], DISPONTE is based on previous work of Lukasiewicz [Luk01b, Luk01a] which combines probabilistic logic programming with conditional constraints. We devote her an extra section because of the existence of the system BUNDLE [RBLZ13] which recently won the best paper award at the Web Reasoning and Rule Systems (RR 2013)[1] conference.

Furthermore, there are some mentionable differences. First, DISPONTE defines exact probabilities p for an axiom a and no minimal and maximal range as in $\mathcal{P}$ – $\mathcal{SROIQ}$. Intuitively, the statement $p :: a$ means that axsiom a is true with probability p and false with probability $1 - p$. Similar to Jaeger et al. [Jae94] they distinguish between conditional probabilities about concepts (e.g. $0.9 :: Car \sqsubseteq HasFourWheels$) and assertional probabilities about concrete instances (e.g. $0.8 :: johns\_car : SportsCar$).

Second, the semantics is based on probabilistic prolog (PROBLOG) [RKT07, FdBT+11]. Compared to classical first-order logic, probabilistic prolog is based

[1] http://rr2013.uni-mannheim.de/

on logic programming and thus inherit the closed-world assumption. In first-order logic, $a \Leftarrow b$ has three models $\{a, b\}$, $\{a, \neg b\}$, and $\{\neg a, \neg b\}$ while the equivalent expression $a : -b$ in logic programming only has one model $\{\neg a, \neg b\}$. The interpretation of the latter case is that there exist no expression which makes b true and, hence, there is no applicable rule that makes a true either.

In Riguzzi et al. [RBLZ12] they extended the DISPONTE semantic with epidemic probabilities. In epidemic probabilities we express the degree of our belief in a certain axiom, while the usual statistical probability is the probability concerning random individuals from certain populations.

Since DISPONTE is closely related to $\mathcal{P}$ – $\mathcal{SROIQ}$ (which was presented in the previous section), we can conclude the semantics of DISPONTE is not comparable with the semantics of log-linear description logics as well.

9.1.3 PR-OWL

Another strand of literature uses (modifications of) Bayesian networks as underlying probabilistic formalism. Since standard Bayeasian networks have limited expressiveness of their attribute-value representation, they can not express sentences like give me all students of an age older than eighteen. It would be necessary to build a new Bayesian network for each instance value change [dCL06].

Thus, the probabilistic ontology language PR-OWL [Cos05, dCL06, LdC12] is based on the multi-entity bayesian network (MEBN) logic [LdC12, Las08]. MEBN specifies a first-order language for modeling probabilistic knowledge bases as parametrized fragments of Bayesian networks. The idea of lifting Bayesian networks to first order logic is related to the idea of combining Markov networks (see Section 2.1 with first-order logic to Markov logic (see Section 2.3).

Before discussing PR-OWL we intuitively introduce multi-entity bayesian networks [Las08, WML$^+$02]. Probabilistic knowledge is expressed in MEBN fragments which is a knowledge structure that represents probabilistic knowledge about a collection of related hypotheses [dCL06]. Like in traditional Bayesian networks, MEBN fragments contain nodes consisting of random variables in a directed graph. These random variables are extended to provide an inner structure so that first-order logic formulas can be represented. A random variable could for instance be $Friend(x, y)$ where x and y stand for any arbitrary person. Each node has a parametrized list of arguments [LdC12]. These arguments are restricted with so called *context nodes*. An example of a context node is $IsPerson(x)$ which defines the possible candidates for variable x. Thus, context nodes are comparable with types in Markov logic (see Section 2.3.1). If a node (like $Friend(x, y)$) has no incoming edges, it is is called an *input node*. A node with incoming edges is called a *resident node*. Each resident node has a distribution which defines the probabilities of its instances given the instances of the parent nodes. Usually, these distributions are defined as static tables. However, since MEBN allows nodes to have an arbitrary number of instances, it is also possible to define functions depending on the input values. We could for instance define a mathematical function

which returns a probability that a person smokes which depends on the number of its smoking friends. Generally, any sentence that can be expressed in first-order logic can also expressed in MEBN as a joint distribution over truth-values of sets of first order logic sentences. For details about the exact syntax and semantics we refer the reader to [Las08, WML$^+$02]. An illustrative example can be found in [LdC12].

As in traditional Bayesian networks, inference requires a query and a set of evidence variables. For inference, MEBN are translated into a situation-specific classical Bayesian network by creating and combining instances of the MEBN fragments. This process can be compared with the procedure of grounding a Markov logic network (see Section 2.3.2). Once we constructed the classical Bayesian network, we are able to perform all the standard inference tasks which are possible for Bayesian networks including marginal inference and maximum a-posteriori inference as described in Section 2.1.2 and Section 2.1.3, respectively.

PR-OWL adds new definition to the standard OWL which enables to create MEBN. Future goals include detailed complexity analysis of sub-languages of PR-OWL and the identification of efficient exact or approximate reasoning algorithms of other domains which can be applied to PR-OWL. Parts of the syntax and the semantics were implemented in the tool UnBBayes-MEBN[dCLC$^+$08] which provide a GUI and a translation of MEBN to classical Bayesian networks. Please refer to Section 9.4 for details.

There seems to be an active community on PR-OWL which recently published PR-OWL 2.0 [CLdC13]. Koller et al. [KLP97] defines the probabilistic logic $P - Classic$ based on Bayesian networks and provide a lifted inference-based algorithm. They prove that for Bayesian networks for which polynomial time reasoning is possible, the inference of their probabilistic logic is also polynomial. Ding and Yun [DP04] propose a set of transformation rules to map OWL ontologies to directed acyclic graphs of a classic Bayesian network and provide an algorithm to compute the corresponding probability tables. This technique has been applied in ontology matching [MNJ05]. Another closely related approach is published in [YC05].

We conclude that it is in principle possible to compute maximum a-posteriori queries out of the situation-specific classical Bayesian network. However, there is a difference in semantics between Bayesian networks and Markov networks. While the graph constructed in Bayesian networks has directed edges, the graph of Markov network has undirected edges. Although there has been some attempts to transfer Bayesian networks to Markov networks and vice versa using essential graphs, they still often result in essential graphs with different meaning [FL07]. Thus, the semantics of the maximum a-posteriori query in log-linear description logics can not be modeled in PR-OWL.

9.2 Possibilistic Description Logics

In this section we explain possibilistic description logics and the corresponding system PossDL [QJPD10]. Early work in possibilistic description logics has been published by [Hol13]. However, since they do not consider reasoning about imprecise concepts and individuals, we summarize from [QJPD10]. Possibilistic description logics is a combination of possibilistic logic and description logics. In the syntax of possibilistic description logics probabilities are attached to any arbitrary description logics axiom x. Thus, a possibilistic axiom is a probability axiom pair $\langle x, p\rangle$ where $p \in (0, 1]$.

For understanding the semantics we first emphasize the semantics of possibilistic logics [DLP94, DLP91]. A distribution of possibility is a so-called membership function Pr that maps elements ϕ and ψ to the unit interval $[0, 1]$ where 0 means non-membership and 1 stands for complete membership. Intuitively, the function Pr represents the degree to which a world is possible. For computing the probabilities of combinations of elements, the following properties hold:

$$Pr(\phi \wedge \psi) = \max(Pr(\phi), Pr(\psi))$$
$$Pr(\phi \vee \psi) = \min(Pr(\phi), Pr(\psi))$$
$$Pr(\neg\phi) = 1 - \min(Pr(\phi), Pr(\psi))$$

The above properties can be inferred from the axioms of probabilistic logic. We encourage the reader to compare them to the properties defined in Section 9.1 and to read [LS08] for mathematical justifications. Thus, probabilistic logic is a superclass of possibilistic logic. Intuitively, in possibilistic logics we are interested if current scenarios are possible whereas in probabilistic logics, we are interested in the particular probability of the scenarios.

We are now in the position to explain the inference tasks which occur when we combine possibilistic logic with description logics. We define $\mathcal{B}_{\geq\alpha} = \{x | \langle x, p\rangle \in \mathcal{B}, w \geq \alpha\}$. Thus, $\mathcal{B}_{\geq\alpha}$ filters those axioms which have a possibility p greater than a certain α. An important question is to compute the inconsistency degree $Inc(\mathcal{B})$ of a knowledge base $\mathcal{B}$. Intuitively, $Inc(\mathcal{B})$ is the maximal degree α where $\mathcal{B}_\alpha$ is barely incoherent. With this degree, we are able to define the following inference tasks for possibilistic description logics [QJPD10]:

- A DL Axiom x is a *plausible* consequence of a knowledge base $\mathcal{B}$ if $\mathcal{B}_{>Inc(\mathcal{B})} \models x$.
- A DL Axiom x is a *possibilistic* consequence of a knowledge base $\mathcal{B}$ to degree α if $\mathcal{B}_{\geq\alpha}$ is consistent, $\mathcal{B}_{\geq\alpha} \models x$, and for all $\beta > \alpha$ we get $\mathcal{B}_{\geq\beta} \not\models x$.
- A possibilistic DL axiom $\langle x, p\rangle$ is a consequence from a knowledge base $\mathcal{B}$ if $p \geq Inc(\mathcal{B})$ and $\mathcal{B}_{\geq p} \models x$.

Example 31 (inspired by [QPJ07]) explains the above definitions and query types in more detail.

Example 31. *Let us assume that we have a DL knowledge base*

$$\mathcal{B} = \{\langle Bird \sqsubseteq CanFly, 0.8\rangle, \langle HasWing \sqsubseteq Bird, 0.95\rangle, \langle HasWing(Tweety), 1\rangle, \langle \neg CanFly(Tweety), 1\rangle\}.$$

Then, $\mathcal{B}_{\geq 0.8} = \mathcal{B}$ *is inconsistent. However,*

$$\mathcal{B}_{\geq 0.95} = \{\langle HasWing \sqsubseteq Bird, 0.95\rangle, \langle HasWing(Tweety), 1\rangle, \langle \neg CanFly(Tweety), 1\rangle\}$$

is consistent. Thus, $Inc(\mathcal{B}) = 0.8$*. Since* $\mathcal{B}_{>0.8} \models Bird(Tweety)$ *we can infer because of the first query, that* $Tweety$ *is plausible to be a* $bird$*. Furthermore,* $Bird(Tweety)$ *is a possibilistic consequence to degree* 0.95 *since* $\mathcal{B}_{\geq 0.95} \models Bird(Tweety)$ *but* $\mathcal{B}_{\geq 1} \not\models Bird(Tweety)$*. According to the last query, the axiom* $\langle Bird(Tweety), 0.9\rangle$ *is a consequence from* $\mathcal{B}$*, because its probability* $0.9 \geq Inc(\mathcal{B}) = 0.8$ *and* $\mathcal{B}_{\geq 0.9} \models Bird(Tweety)$*.*

After we understood the query types of possibilistic description logics, it becomes clear that they are very different from the most probable coherent world query asked in log-linear description logics. In the most-probable coherent world we often prefer many axioms with low weights over few axioms with high weights if their overall sum is higher. This is illustrated in Example 22. In possibilistic description logics, we cut every axiom which is under a certain probability α. Thus, we can not compare those two queries.

9.3 Fuzzy Description Logics

Fuzzy description logics is based on fuzzy logic. Fuzzy-logic, which is a subcategory of so-called many-valued logics, defines a more refined range for truth and false values. In particular, the usual true/false convention is expressed as a degree within $[0, 1]$ of being true/false. A frequently used example is the expression of a *tall man*. Since the height of a man is graded, we might prefer to say that a man is to 80% tall rather than defining a fix boarder at which a man is tall or not tall [LS08].

Fuzzy formulas often have the form $\phi \geq l$ or $\phi \leq u$ where $l, u \in [0, 1]$ which encode the degree of truth of ϕ is at least l or at most u [Häh01]. Thus, the fuzzy interpretation $\mathcal{I}$ maps a basic true/false value to the interval $[0, 1]$ and defines combination functions $\mathcal{I}(\phi \wedge \psi)$ and $\mathcal{I}(\phi \vee \psi)$, implication functions $\mathcal{I}(\phi \Rightarrow \psi)$, and negation functions $\mathcal{I}(\neg\phi)$. In literature, several different functions have been proposed which all satisfy different properties. Table 9.1 lists the most popular implementations. For details which properties are satisfied with which implementations, we refer the reader to [Häh01, Nov06].

In literature, there exist several approaches which combine description logics with fuzzy logic to fuzzy description logicsfuzzy description logics. We refer

	Lukasiewicz logic	Gödel logic	Product logic	Zadeh logic
$\mathcal{I}(\phi \wedge \psi)$	$max(\phi + \psi - 1, 0)$	$min(\phi, \psi)$	$\phi \times \psi$	$min(\phi, \psi)$
$\mathcal{I}(\phi \vee \psi)$	$min(\phi + \psi, 1)$	$max(\phi, \psi)$	$\phi + \psi - \phi \times \psi$	$max(\phi, \psi)$
$\mathcal{I}(\phi \Rightarrow \psi)$	$min(1 - \phi + \psi, 1)$	$\begin{cases} 1 & if \phi \leq \psi \\ \psi & otherwise \end{cases}$	$min(1, \psi/\phi)$	$max(1 - \phi, \psi)$
$\mathcal{I}(\neg\phi)$	$1 - \phi$	$\begin{cases} 1 & if \phi = 0 \\ 0 & otherwise \end{cases}$	$\begin{cases} 1 & if \phi = 0 \\ 0 & otherwise \end{cases}$	$1 - \phi$

Table 9.1: Combination functions of various fuzzy logics [Häh01, LS08].

to [LS08] for a survey. Newer approaches include Straccia [Str05a] in which he introduces a fuzzy logic with concrete domains in which fuzzy modifiers are allowed and the reasoning algorithm is based on a mixture of completion rules and bounded mixed integer programming. Bobillo et al. [BDGRS09] introduce a fuzzy logic under the Gödel semantics. More theoretical work discuss the question whether description logics with general concept inclusion axioms are decidable [BP11, BP12] by giving strong indications for undecidability. Simou et al. [SMSS10] present optimization techniques like degree normalization, assertional box partitioning, and optimized greatest lower bounds that have the potential to improve the performance of fuzzy-DL systems.

In the following, we briefly report *fuzzy* $\mathcal{SHOIN}$**(D)** [Str05b, Str06] since this approach is comparable expressive with respect to the description logic language, the allowed fuzzy constructs, and the underlying fuzzy logics.

The syntax of *fuzzy* $\mathcal{SHOIN}$**(D)** allows to define fuzzy datatypes. We can for example define a fuzzy concept $Young$ which expresses the degree of youthfulness of a person's age. We could for example use a left-shoulder function expressing that a person is young until she is 10 years old, then linearly decreasing the degree of youth until 30. Having an age older than 30 she is defined as not being young any more. With that knowledge, we then can define further fuzzy concepts like $YoungPerson \equiv Person \sqcap \exists age.Young$. Additionally, fuzzy modifiers like $very$ apply to fuzzy sets to change their membership functions. To that end, we assign a function to each fuzzy modifiers like $f_{very} = x^2$. With this knowledge we could define a $VeryYoungPerson$ with $Person \sqcap \exists age.very(Young)$. We refer to [Str05b] for further functions specifying fuzzy set membership degrees and formal introduction of syntax and semantics of *fuzzy* $\mathcal{SHOIN}$**(D)**.

Reasoning tasks in fuzzy description logics include the standard problems like satisfiablity of fuzzy knowledge bases, deciding the satisfiability of concepts relative to fuzzy knowledge bases, and deciding logical consequences of fuzzy axioms from fuzzy knowledge bases. Despite these reasoning tasks there exist two additional queries. First, the best truth value bound query computes the best lower and upper truth value bounds of an axiom a. Second, the best satisfiability bound of a concept c intuitively determines the maximal degree of truth that the concept c may have over all individuals [Str05b, LS08]. These queries can be solved with mixed integer linear programming [SB07].

Log-linear description logics does not allow a degree of truth value since it is

based on classical first-order logic. Thus, syntax and semantics of fuzzy description logics are not comparable with log-linear description logics. However, in future work we might integrate fuzzy concepts in log-linear description logics.

9.4 Systems

To the best of our knowledge, there exist six other systems next to our system ELOG which combine description logics with some kind of uncertainty. Two out of these six systems (namely BUNDLE and POSSDL) are not publicly available any more. All systems except INCERTO are based on one of the probabilistic, possibilistic, and fuzzy logics that we described previously. Since we already argued in the respective sections above, that the semantics of these logics are not comparable with the semantics of log-linear description logics, we conclude that we can not compare our system ELOG with them. The system INCERTO also utilizes Markov logic as background logic. However, they only support marginal inference and their semantics is different from those of log-linear description logics.

For details, we refer the reader to the respective sections about the systems below.

PRONTO

Klinov [Kli11] developed in his dissertation the probabilistic reasoner PRONTO[2]. Afterwards, Klinov starts working at the company CLARKPARSIA where he did further developments on PRONTO. When Klinov left CLARKPARSIA they decided to skip the further development. Since PRONTO is not longer under an open-source license it is not available for download any more.

PRONTO [Kli11] is an implementation of $\mathcal{P}-\mathcal{SROIQ}$ (refer to Section 9.1.1) and supports all the inference types mentioned in Section 9.1.1. For inference, PELLET[3] is used.

BUNDLE

The reasoner BUNDLE [RBLZ13, BLRA11][4] stands for 'Binary decision diagrams for uncertain reasoning on description logic theories' and is based on the DISPONTE semantics introduced in Section 9.1.2. For inference, BUNDLE iteratively finds each explanation and then builds a binary decision diagram for computing the probability of the query. In particular, it first queries for a single explanation. Then, it iteratively removes each axiom in this explanation and asks again for another single explanation in the remaining set of axioms. This process is continued until no more explanations are found. Binary decision diagrams, finally, allows an efficient computation of the probabilities which directly result from the explanations.

[2] http://weblog.clarkparsia.com/2007/09/27/introducing-pronto

[3] http://clarkparsia.com/pellet/

[4] https://sites.google.com/a/unife.it/ml/bundle

Internally, BUNDLE exploits the description logic reasoner PELLET to query explanations. Currently, the possible queries are restricted to retrieving the probability of the subsumption of two concepts, concept and property assertions, and the satisfiability of all concepts or specific concepts. Furthermore, consistency can be checked and a hierarchy can be computed.

UNBBAYES-MEBN

The focus of the tool UNBBAYES-MEBN[dCLC+08][5] lies on its GUI interface, which allows to graphically model most of the functionality of PR-OWL (see Section 9.1.3). It allows users to model probabilistic knowledge bases without having to rely on a deep knowledge in PR-OWL. For integrating logical functions UNBBAYES-MEBN uses the knowledge representation tool POWERLOOM[6] providing multiple build-in deductive reasoning capabilities. The support of the construction of situation-specific classical Bayesian networks enables UNBBAYES-MEBN to perform reasoning tasks. However, in the current implementation queries are restricted to a single random variable instance which is not allowed to have any evidence below it [dCLC+08].

POSSDL

POSSDL [QJPD10] is a reasoner for possibilistic description logics (refer to Section 9.2), which was developed in 2011 from Jeff Pan. According to [QJPD10] it has been implemented as an extension of the Neon Toolkit[7]. However, it is not available for download.

The reasoner supports the computation of the inconsistency degree as well as instance and subsumption checking with a necessity degree. Despite the fact that they use the PELLET reasoner for computing the query answers, no further details about their algorithms have been published.

FIRE

The focus of the fuzzy reasoning engine FIRE [SSSK06, SSS13] is on efficient storage and querying of fuzzy description logics as described in Section 9.3. For storing a triple store is used. Possible queries implemented in FIRE are satisfiability checking, subsumption and entailment of concepts and axioms, and determination of the best lower and upper truth value bounds of an axiom. For these inference tasks, FIRE implements the tableau algorithm proposed by [SSP+07].

[5] http://unbbayes.sourceforge.net/changes-report.html
[6] http://www.isi.edu/isd/LOOM/PowerLoom/
[7] http://neon-toolkit.org/wiki/MainPage

INCERTO

To the best of our knowledge, INCERTO[8] is the only reasoner which uses Markov logic for probabilistic description logics. They directly translate the description logic axioms to Markov logic so that concepts correspond to unary predicates, roles correspond to binary predicates, and individuals to constants. The description logic axioms are translated to first-order logic formulas[9]. Example 32 illustrates the translation.

Example 32. *Let $\mathcal{O}_1$ be the ontology which we defined in Example 21. When we translate it according to the semantic used in* INCERTO *we obtain the following first-order formulas:*

(1)	$Cat \sqsubseteq Animal$	$Cat(x) \Rightarrow Animal(x)$
(2)	$Cat \sqcap Brand \sqsubseteq \bot$	$Cat(x) \Rightarrow \neg Brand(x)$
(3)	$\langle Jaguar \sqsubseteq Cat, 0.5\rangle$	$\langle Jaguar(x) \Rightarrow Cat(x), 0.5\rangle$
(4)	$\langle Jaguar \sqsubseteq Animal, 0.9\rangle$	$\langle Jaguar(x) \Rightarrow Animal(x), 0.9\rangle$
(5)	$\langle Jaguar \sqsubseteq Brand, 1.2\rangle$	$\langle Jaguar(x) \Rightarrow Brand(x), 1.2\rangle$

Their main objective is to learn the weights of axioms through the analysis of individuals. Furthermore, they provide exact and approximate marginal inference. However, computing the most probable coherent ontology is not supported. In fact, the translation presented in Example 32 does not allow this computation, since no incoherency is derived. Only when dummy individual assertions are inserted it is possible to derive a contradiction. On a first glimpse, creating one individual c for every concept C and adding the concept assertions $C(c)$ leads to a similar semantic than log-linear description logic. However, appearances are deceptive as shown in Example 33.

Example 33. *For illustrating the difference in the semantic, let us modify the weight of the following axiom to*

$$\langle Jaguar \sqsubseteq Animal, 0.5\rangle$$

leaving the other axioms unchanged.

If we now compute the most probable coherent world with log-linear description logics, we obtain the result

(1)	$Cat \sqsubseteq Animal$
(2)	$Cat \sqcap Brand \sqsubseteq \bot$
(5)	$Jaguar \sqsubseteq Brand$

since the weight of axiom $Jaguar \sqsubseteq Brand$ (= 1.2) is higher than the sum of the weights of $Jaguar \sqsubseteq Animal$ and $Jaguar \sqsubseteq Cat$ (= 0.5 + 0.5 = 1.0).

[8] `https://code.google.com/p/incerto/`

[9] Their exact translations are available at `http://incerto.googlecode.com/files/TranslationOWL2FOL.pdf`

Let us now introduce for every concept $Animal$, $Brand$, Cat, and $Jaguar$ the corresponding concept assertions $Animal(a)$, $Brand(b)$, $Cat(c)$, and $Jaguar(j)$. As before, we have to choose either axiom (3) and (4) or axiom (5). This time, however, the sum of the weights derived from axiom (3) and (4) is $0.5 + 2 \cdot 0.5 = 1.5$, since we infer $Animal(c)$ from $Cat(c)$. Thus, we count the weight of $Jaguar \sqsubseteq Animal$ twice - one time due to $Animal(c)$ and one time due to $Animal(a)$. Consequently, we receive a different result for the translation performed by INCERTO*:*

(1)	$Cat \sqsubseteq Animal$
(2)	$Cat \sqcap Brand \sqsubseteq \bot$
(3)	$Jaguar \sqsubseteq Cat$
(4)	$Jaguar \sqsubseteq Animal$

The difference in semantics illustrated in Example 33 becomes worse when we consider object properties and their relation to concepts like domain and range restrictions. To the best of our knowledge, there exist no feasible algorithm which leads to the identical syntax. Consequently, we are not able to compare ELOG with INCERTO.

Chapter 10

Experiments

To complement the presented theory we also assessed the practicality of the reasoning algorithms. After all, the development of the theory was motivated primarily by the need for algorithms that, given a set of axioms with confidence values, compute a most probable *coherent* ontology.

Before we report about experimental results, we present details about our log-linear description logic reasoner ELOG, which implements the MAP-query for log-linear description logics, in Section 10.1. In the experiments, we focus on answering the last research questions from Section 1.2.2:

Q6 *Can we experimentally verify that a solution's quality increase with increasing expressivity and that optimal solving strategies result in higher quality solutions than approximate solving strategies?*

The question asks for an increase in quality. However, its first part asks if the quality increases if we increase the expressivity while its second part asks if optimal solving strategies lead to higher quality than than approximate strategies.

We address the research question by conducting experiments in the area of ontology learning and ontology matching since they are the targeted application areas of log-linear description logics. We refer the interested reader to Section 10.2 for details about (possible) applications.

This chapter provides more exhaustive experiments than our publications [NN11, NNS11]. Some text passages, pictures, and results especially for the ontology learning benchmarks are taken from our publications [NN11, NNS11].

10.1 Log-Linear DL Reasoner ELOG

We have implemented the computation of the most-probable coherent ontology in the log-linear description logics reasoner ELOG. In this section we will dive a little bit deeper into implementation details.

As for ROCKIT, we also provide a similar easy-to-use web-service for ELOG

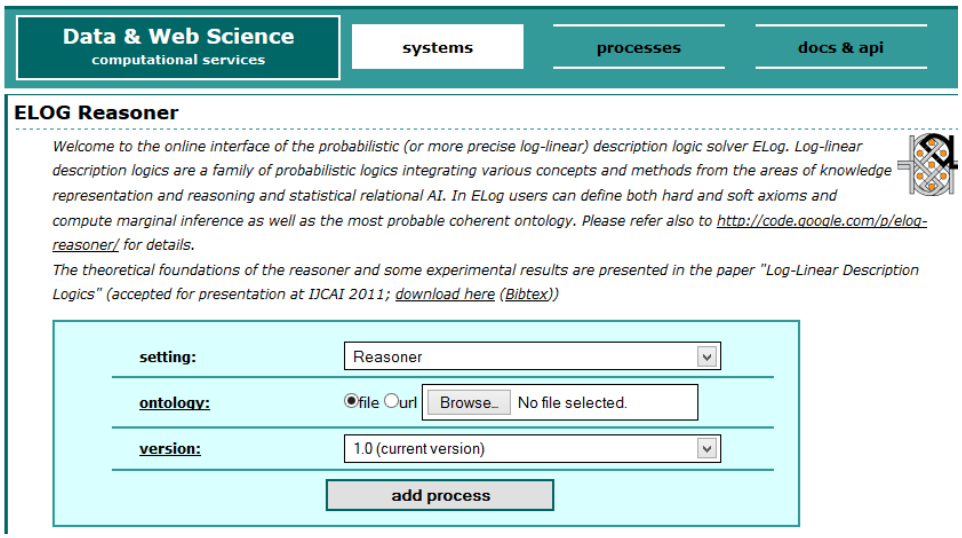

Figure 10.1: Screenshot of the online web-interface of ELOG. Users can easily solve their log-linear ontologies without any configuration and installation effort.

in addition to the source-code, a documentation, and installation instructions[1]. Figure 10.1 provides a screenshot of the user interface. Furthermore, programmers can integrate the MLN engine in their application via existing REST interfaces.

Computing the most probable coherent ontology with ELOG is implemented for $\mathcal{EL}^{++}$. As mentioned earlier, it is straight forward to implement more expressive description logics like introduced in Section 8.1. For parsing ontologies, we implement the OWLAPI[2]. Thus, ELOG supports every ontology format which is also supported by the OWLAPI including the XML, functional, and turtle syntax. Weights are encoded by attaching the annotation property `confidence`, having the weight as value, to the respective axiom.

All axioms that are outside of the scope of $\mathcal{EL}^{++}$ are not considered. Normalization is implemented in Java utilizing some efficient graph data structures from the JGRAPH[3] library. In particular, the graph pre-materialize subsumption trees. Consequently, less cutting plane inference loops have to be performed and runtime decreases. The normalized ontology is then translated into a Markov logic network as described in Section 8.3. As Markov logic solver, we integrate our ROCKIT solver which implements the theory of Part I of this thesis. After the execution of ROCKIT, ELOG translates the retrieved MAP state back to ontology axioms and returns a *materialized* OWL ontology.

Please note that ELOG also implements alternative algorithms for computing

[1] The user interface is available at `http://executor.informatik.uni-mannheim.de/systems/elog/`.

[2] `http://owlapi.sourceforge.net/`

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

the most probable world like a greedy algorithm. Furthermore, there are possibilities to compute marginal probabilities as well.

10.2 Applications

The maximum a-posteriori query for log-linear description logics can be applied in areas where we have given uncertain axioms and aim to find the most probable coherent ontology. Intuitively, it returns the ontology where the least (weighted) amount of axioms are thrown away such that the resulting ontology is just coherent. Research areas where those ontologies are constructed are ontology learning and ontology matching. Furthermore, log-linear description logics have recently been applied in the area of activity recognition. This section gives a brief overview about these areas and briefly illustrates how log-linear DL can be applied in each of them.

10.2.1 Ontology Learning and Ontology Debugging

In ontology learning, (semi-)automatic techniques are applied on either structured or unstructured data to learn new axiom types. Several workshops have been taken place in this area [Bre06]. For extracting ontologies from unstructured data, there exist two general framework. The earlier one called TextToOnto and was build by Maedche and Staab [MS04]. Later, Cimiano and Völker published the system Text2Onto [CV05]. A comparison between the two systems can be found in [CMSV09]. Johanna Voelker published several additional articles in this field like learning disjointness axioms [VVSH07] and enriching ontologies with axioms of higher complexity [VN11]. For a general overview of ontology learning approaches we forward the reader to the following surveys: [Zho07] gives a more general overview of ontology learning, Ciminao et al. [CMSV09] provides a comprehensive and concise overview, and [Bre06, WLB12] survey methods to learn ontologies from text.

The ontology learning community has developed and applied numerous machine learning and data mining algorithms to generate confidence values for DL axioms. However, most of these confidence values have no clearly defined semantics. Confidence values based on lexical similarity measures, for instance, are in widespread use while more sophisticated algorithms that generate actual probabilities make often naïve assumptions about the dependencies of the underlying probability distribution. Hence, formalisms are needed that incorporate these various types of confidence values in order to compute most probable ontologies while utilizing the logical concepts of coherency and consistency.

With help of the MAP query of log-linear description logics, we can help ontology learning scientists to repair their ontologies while guarantying to maximize the sum of the remaining learned axioms. Intuitively, ELOG returns as many axioms with possibly high weights so that the output ontology remains coherent. Since almost all approaches in ontology learning create axioms attached with confidence

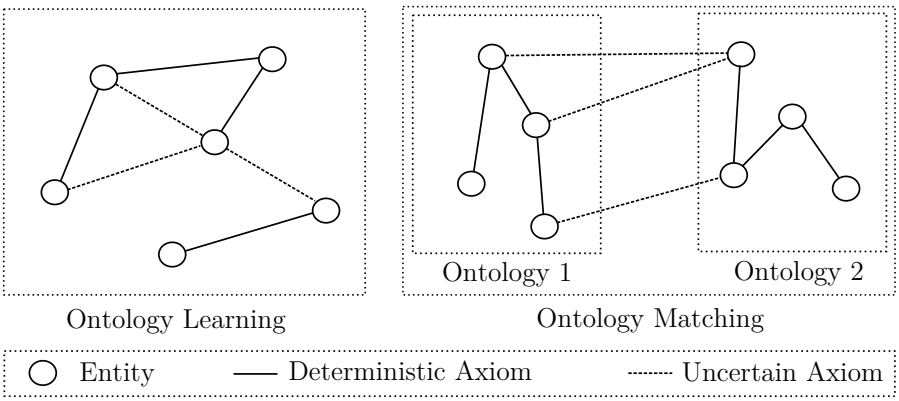

Figure 10.2: Difference between ontology matching and ontology learning.

values, we can model them directly as uncertain CBox $\mathcal{C}^{\mathsf{U}}$ in log-linear description logics. If there exist a deterministic ontology as background knowledge, we can integrate its axioms by mapping them to the deterministic CBox $\mathcal{C}^{\mathsf{D}}$.

This task is related to ontology debugging [PSK05]. In ontology debugging, the task is to detect and correct incoherencies and inconsistencies within ontologies. However, according to Stuckenschmidt [Stu08] ontology debugging systems in and before 2008 were mostly theoretical. In practical applications they failed in well-foundness, robustness, or performance with respect to run-time. For details, we refer the interested reader to [Stu08]. More recent ontology debugging systems either focus on interactive debugging with the goal to minimize user interaction [SFFR12, RSFF12] or examine the theoretical properties of ontology debugging without providing scalable implementations [T$^+$13].

10.2.2 Ontology Matching

In ontology matching we take two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ as input and find correspondences between their entities (like concepts or roles). These correspondences often have a weight (called confidence value) attached. A set of correspondences $\mathcal{M}$ is called alignment. An alignment is called coherent if $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$ is coherent. Please refer to [ES07] for a more formal and exhaustive introduction to ontology matching.

Figure 10.2 illustrates the difference between ontology learning and ontology matching with respect of connections between axioms. In ontology learning, uncertain and deterministic axioms can occur between any kind of entities. In ontology matching, deterministic axioms only connect entities within one ontology while uncertain axioms only connect entities between ontologies. Thus, the application of ontology matching is a specialized application of ontology learning.

Furthermore, matching systems and existing benchmarks often focus on find-

ing equivalence correspondences (axioms of type $A \equiv B$), which semantically means they aim to identify classes, properties, or instances which belong to the same real-world entities. There is a central initiative, called the ontology alignment evaluation initiative (OAEI) [EMS+11, SE13], which has been evaluating and comparing different matching systems for nine years now. The goal of the initiative is to compare alignment systems on the same basis and point out their strength and weaknesses.

Fewer approaches exist for complex matching. Here, the task is to detect more complex correspondences which include for example more than two entities or relationships between different entities like classes and roles. In Ritze et. al [RMSZS09, RVMSZ10] they extracted complex axioms based on existing structural and lexical patterns and applied techniques from natural language processing on complex matching tasks, respectively. Closely related is the work from Dhamankar et al. [DLD+04] in which they extract complex matches in database schema.

Over the past years, the importance of producing coherent alignments has been shown and algorithms have been developed for making alignments coherent [MST07, Mei11] and, additionally, for awarding structural similarities with the introduction of soft constraints [NMS10]. We implemented those strategies in the CODI matcher, which was able to gain competitive results at the OAEI campaign [HSNM11, NN10]. Additionally, we applied similar ideas in the area of instance matching [NNMS10].

Log-linear description logics guarantee for a specific description logic, that the alignment is coherent and that, intuitively speaking, we drop as few other (weighted) alignments as possible. To that end, the two merged input ontologies are taken as the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ while the created alignments with the confidence values are encoded as the uncertain CBox $\mathcal{C}^{\mathsf{U}}$.

In ontology matching, two specialized alignment repair systems called ALCOMO [Mei11, MST07][4] and LOGMAP [JRG11][5] have been developed. Repairing alignments is the task to take a given alignment and return an (almost) coherent alignment while maximizing the sum of the confidence values. This is equivalent to the task of finding the most-probable coherent ontology applied on the ontology $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$.

However, these systems are restricted to the application in the area of ontology matching since they exploit the fact that uncertain knowledge only occurs *between* ontologies. Thus, they can not cope with the application areas ontology learning and activity recognition. In the following sections we discuss their algorithms in further detail since we compare ELOG against them in our experimental section. The explanations of the systems are mainly summarized from [JRMGH13].

[4] http://web.informatik.uni-mannheim.de/alcomo/
[5] http://www.cs.ox.ac.uk/isg/projects/LogMap/

Alcomo

The basic idea of ALCOMO is to avoid computing all minimal incoherent alignment subsets, but to first classify both ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ separately using an OWL reasoner. After classification, ALCOMO solves two interconnected problems at the same time: (i) the reasoning problem to detect if a mapping is incoherent and (ii) the optimization problem of taking confidences into account in an appropriate way. In both problems ALCOMO implements an optimal and an approximate (greedy) approach. In our experiments we refer to ALCOMO *optimal*, if both problems are solved in an optimal way and to ALCOMO *greedy*, if in both tasks the approximate variant is chosen.

The reasoning problem checks if a given set of alignments $\mathcal{M}$ is coherence. In the approximate approach a limited number of patterns are checked which can be done in quadratic time with respect to the alignment size. These patterns take subsumption and equivalence alignments between concepts and properties into account. Let us assume that we have for instance two correspondences $A_1 \equiv C_2 \in \mathcal{M}$ and $B_1 \equiv D_2 \in \mathcal{M}$ where $A_1, B_1 \in \mathcal{O}_1$ and $C_2, D_2 \in \mathcal{O}_2$. Then, one pattern of ALCOMO checks if $\mathcal{O} \models A_1 \sqsubseteq B_1$ and $\mathcal{O} \models C_2 \sqsubseteq \neg D_2$. Please note, that due to the classification at the beginning, these entailments can be checked in constant time. If this holds, it follows that $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M} \models C_2 \sqsubseteq \bot$ and thus that the alignment $\mathcal{M}$ is incoherent. However, it is known that these rules do not cover all types of possible incoherencies and thus do not ensure coherency. It is also not known for which underlying description logic full alignment coherency is ensured [Mei11].

The complete reasoning approach uses a description logics reasoner as a black box and is based on classical black-box approaches for repairing ontologies as described in [SHCvH07]. The reasoner is used to compute so-called minimal unsatisfiability-preserving sub-TBoxes (MUPS) which are, descriptively speaking, minimal conflict sets. We refer the reader to [Mei11, SHCvH07] for details. In order to reduce the number of reasoner calls, ALCOMO uses the result of the approximate reasoning algorithm as starting point. If it is inconsistent, it is further reduced. This is permissible because it is proven that the results of the approximate technique always entails the results of the complete reasoning techniques [Mei11].

With respect to the optimization problem, ALCOMO implements a global optimal diagnosis which maximizes the sum of the given confidence values. For detecting this optimum, ALCOMO implements an A*-search [HNR68]. The algorithm starts with a complete alignment in the root node and reduces one correspondence from the alignment for each successor step. The candidate for removal is detected with the reasoning techniques sketched above. Optimizations are made by storing the MUPS that have been previously found with the approximate reasoning techniques and by using a heuristic function to estimate the remaining costs for a search node. Again, we refer the interested reader to [Mei11] for details. For large search problems, however, this global optimal diagnosis is not feasible.

The second algorithm implemented to tackle the optimization problem is a greedy algorithm. In this algorithm we initially start with an empty alignment $\mathcal{M}'$

and a set $\mathcal{M}$ in which the correspondences are ordered according to their confidence value - starting with the highest one. Then we iterate over $\mathcal{M}$ and put the correspondence $c \in \mathcal{M}$ to the alignment $\mathcal{M}'$ if the alignment $\mathcal{M}'$ is still coherent. Else, we take the next correspondence. Coherence checking can again be performed with either the approximate or the complete reasoning approach.

LogMap

The basic idea of LOGMAP is to translate the ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ into a set $\mathcal{P}$ of Horn clauses and apply the linear Dowling-Gallier algorithm for propositional Horn satisfiability [DG84] multiple times for repairing alignments.

The translation is restricted to only three axiom types. Subsumption axioms of form $A \sqsubseteq B$ are translated to the rule $A \Rightarrow B$, disjointness axioms of form $A \sqcap B \sqsubseteq \bot$ are translated to $A \wedge B \Rightarrow False$, and axioms of type $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B$ are translated to $A_1 \wedge \ldots \wedge A_n \Rightarrow B$. All equivalence correspondences in alignment $\mathcal{M}$ are split into two subsumption alignments, translated to horn clauses, and put into a set $\mathcal{M}'$. The following algorithm starts with the full alignment.

The repair algorithm of LOGMAP goes through every variable v (corresponding to ontological entities) in $\mathcal{P}$. As an optimization, the variables are ordered in a preprocessing step so that a variable C comes before D whenever $D \sqsubseteq C$ in $\mathcal{O}_1$ or in $\mathcal{O}_2$. The motivation behind this ordering is that if a class is unsatisfiable, its subclass is also unsatisfiable. Thus, the class needs to be repaired first, before repairing its subclasses.

For each variable v LOGMAP checks with the linear Dowling-Gallier algorithm if $\mathcal{P} \cup \mathcal{M}' \cup \{True \Rightarrow v\}$ is satisfiable. If it is satisfiable, LOGMAP proceeds with the next variable v. If it is not satisfiable, LOGMAP operates on an overestimated set of conflict mappings $\mathcal{M}_\bot$ which is returned by the slightly modified Dowling-Gallier algorithm. Instead of computing a diagnosis for the unsatisfiable variable v, LOGMAP aims to find the repairs of the smallest size. To that end, the algorithm goes through all subsets $\mathcal{R}_v$ of the conflict mappings $\mathcal{M}_\bot$ of $size = 1$. It then checks with the Dowling-Gallier algorithm if $\mathcal{P} \cup \mathcal{M}' \cup \{True \Rightarrow v\}$
$\mathcal{R}_v$ is satisfiable. If yes, it removes $\mathcal{R}_v$ from the mappings $\mathcal{M}'$. If more than one $\mathcal{R}_v$ with $size = 1$ exists, it takes the one for which the sum of the confidence values is minimal. If no $\mathcal{R}_v$ with $size = 1$ is found it increases the size by 1 and checks all subsets again. Since the size of $\mathcal{M}_\bot$ and $\mathcal{R}_v$ are manageable in practice, the complexity of this last procedure is not critical.

The algorithm is approximate since it does not consider many axiom types and thus the encoding of the ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ is incomplete. Properties are, for example, completely ignored. Compared to log-linear description logics implemented in $\mathcal{EL}^{++}$, LOGMAP only supports the rules F_1-F_4 and F_9 of Table 8.5 and only the two axiom types sub and int of the six axiom types from Definition 4.

10.2.3 Activity Recognition

Recently, log-linear description logics has been applied in the field of activity recognition [HRN+12, HRS13] in which the current human activity has to be predicted on the basis of light-weight wearable and environmental sensors. In their work, Helaoui et al. created an ontology representing a multilevel activity recognition framework. The lowest level is called atomic gesture and equals the data which can directly be derived from the current sensor data (like $ReachMilk$ and $MoveMilk$). The next level are the manipulative gestures (like $FetchMilk$) which are derived from several atomic gestures. Afterwards, simple activities (like $PutawayMilk$) are again derived from several manipulative gestures and, finally, several simple activities lead to one complex activity (like $CleaningUp$). The levels are connected with several uncertain axioms represented in the uncertain CBox $\mathcal{C}^{\mathsf{U}}$ and some hard disjointness axioms mapped to the deterministic CBox $\mathcal{C}^{\mathsf{D}}$. An example of one uncertain axiom connecting manipulative gestures with atomic gestures is

$$\langle FetchMilk \sqsupseteq ManipulativeGesture \sqcap \exists hasActor(Person \sqcap \exists hasAtomicGesture.ReachMilk), 0.8\rangle .$$

Finally, they utilize our ELOG reasoner to infer the next level activities until the last level is reached.

10.3 Benchmarks

With our experiments we cover the application areas ontology matching (see Section 10.2.2) and ontology learning (see Section 10.2.1). For experiments with log-linear description logics on activity recognition we refer the interested reader to [HRS13].

10.3.1 Ontology Matching

The benchmarks in ontology matching originate from the ontology alignment evaluation initiative (OAEI). We selected the CONFERENCE (see Section 10.3.1) and the LARGE BIOMED (see Section 10.3.1) benchmarks due to the existence of disjointness axioms and due to the fact that these benchmarks were not artificially created.

CONFERENCE Benchmark

As the name implies, the CONFERENCE benchmark consists of 15 ontologies which describe different scientific conferences. These ontologies have been initially created at a workshop at the University of Economics in Prague and have been improved later by Meilicke and Šváb. For details we refer to [ŠSB+05]. Reference alignments have been created between all pairs of seven ontologies, resulting in

	cmt	conference	confof	edas	ekaw	iasted	sigkdd
classes	36	60	38	104	77	140	49
properties	59	64	36	50	33	41	28
subsumption	25	49	33	84	71	132	41
+ disjointness	52	63	76	491	145	133	41
+ domain and range restrictions	149	149	100	543	184	193	73
+ all other $\mathcal{EL}^{++}$ axioms	263	331	293	865	309	505	186
every axiom	318	408	335	903	341	539	193

Table 10.1: Number of classes, properties, and deterministic axioms in the CONFERENCE ontologies.

	conference	confof	edas	ekaw	iasted	sigkdd
cmt	51	31	45	36	20	29
conference	-	36	61	60	41	42
confof	-	-	49	41	27	25
edas	-	-	-	56	49	41
ekaw	-	-	-	-	51	29
iasted	-	-	-	-	-	60

Table 10.2: Number of weighted equivalent axioms in the merged CONFERENCE alignments if no threshold is applied. The total number is 880.

21 pairs for which a reference alignment exists. The number of classes, properties, and axiom types of these seven ontologies are displayed in Table 10.1. The number of axiom types are ordered according to increasing expressiveness. First, only subsumption axioms of form $A \sqsubseteq B$ are counted. Then, we add the number of disjointness axioms of type $A \sqcap B \sqsubseteq \bot$, domain and range restrictions, all remaining $\mathcal{EL}^{++}$ axioms, and, finally, every existing axiom. This expressiveness staggering is needed for our experiments in Section 10.5.1.

Inspired by [Mei11], we aggregated all matcher results of the 2013 OAEI campaign, which have been above the string equality boarder[6]. Thus, we aggregated the results of YAM++, LOGMAP, AML, ODGOMS1, STRINGSAUTO, SERVOMAP, MAPSSS, HERTUDA, WIKIMATCH, WESEE-MATCH, IAMA, HOTMATCH, CIDER and its variants. Details about most of these systems as well as references to their respective publications can be found at [AEE+12].

[6] http://oaei.ontologymatching.org/2013/conference/eval.html

The computation of the aggregation of the alignment results is similar to Meilicke [Mei11]. For each of the 21 pairs, we union the alignments $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of each matching system to one alignment

$$\mathcal{A} = \bigcup_i \mathcal{A}_i.$$

To that end, we first span the confidence values w of each correspondence $\langle w, a\rangle$ in alignment $\mathcal{A}_i$ to the range of $(0, 1]$ with $w = 0$ if $\langle w, a\rangle \notin \mathcal{A}_i$. In that way, we ensure that all matching results are weighted equally. Afterwards, we union all alignments by taking the average of all confidence values

$$\langle w, a\rangle \in \mathcal{A} = \langle \sum_i \{w_i | \langle w_i, a\rangle \in \mathcal{A}_i\}/n, a\rangle.$$

Note that all alignments a are equivalent axioms between classes or properties. Table 10.2 lists the number of weighted equivalent axioms with $w > 0$ in the merged alignment if no threshold is applied.

Since log-linear description logics require a CBox $\mathcal{C} = (\mathcal{C}^{\mathsf{D}}, \mathcal{C}^{\mathsf{U}})$, we merge two ontologies to the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ and set the aggregated alignment $\mathcal{A}$ as uncertain CBox $\mathcal{C}^{\mathsf{U}}$. The left part of Figure 10.3 visualizes the structure of the CONFERENCE benchmark and, exemplarily, the merge process for the EKAW and EDAS ontology. Thus, we obtain 21 different CBoxes in total.

LARGE BIOMED Benchmark

Since the ontologies in the CONFERENCE benchmark are relatively small, we additionally perform experiments on the LARGE BIOMED benchmark[8]. The benchmark consists of the three datasets Foundational Model of Anatomy (FMA)[9], Na-

[8] We took the small fragments of the benchmark.

[9] http://sig.biostr.washington.edu/projects/fm/

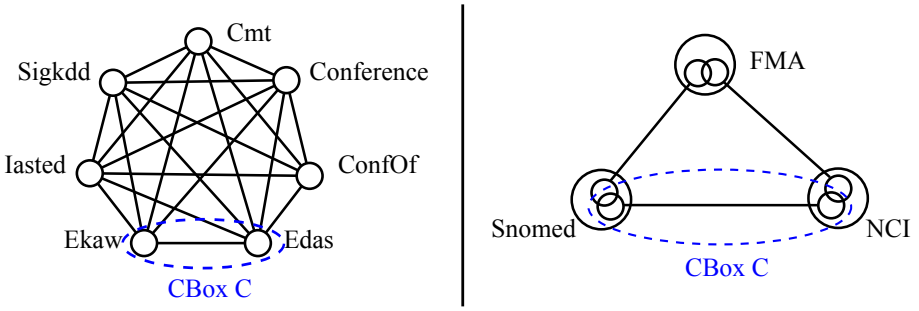

Figure 10.3: Structure of the CONFERENCE (left) and LARGE BIOMED (right) ontology matching benchmarks (inspired by [Mei11]).

tional Cancer Institute Thesaurus (NCI)[10], and SNOMED clinical terms[11]. Being semantically rich and consisting of thousands of classes the benchmark is one of the toughest in the OAEI campaign. In 2012 only six out of 21 systems were able to compute results for each of the three combinations. 11 systems could compute at least one combination.

The properties of the benchmark are summarized in Table 10.3. Since the benchmark only contains the overlapping fragments, we obtain different ontologies depending on the respective matching canidate. There exist, for example, two fragments of the FMA ontologies - one containing the axioms overlapping with the NCI ontology and one containing the axioms overlapping with the SNOMED ontology. For details about the axiom expressiveness levels, please study their description in the CONFERENCE benchmark.

Although reference alignments between each of the datasets exist, we decided not to use it for our evaluation since they have been created automatically by utilizing the Unified Medical Language System (UMLS)[12] as dictionary, *harmonizing* the output of different matching systems, and repairing the reference alignment with LOGMAP[13]. Consequently, the resulting *silver* standard is build such that best results will be achieved with the LOGMAP repair technique. Optimal techniques will always result in lower f-scores.

As for the conference benchmark, we union the results from the different matching systems. We restricted our alignment to Systems which were able to compute results for each of the three combinations. This was the case for the systems

[10] http://ncit.nci.nih.gov/

[11] http://www.ihtsdo.org/index.php?id=545

[12] http://www.nlm.nih.gov/research/umls/

[13] For details we refer to http://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/2012/harmo2012.html

	FMA for NCI	FMA for SNOMED	NCI for FMA	NCI for SNOMED	SNOMED for FMA	SNOMED for NCI
classes	3696	10157	6488	23958	13412	51128
properties	24	24	63	82	18	51
subsumption	3693	10154	4917	18946	16287	31299
+ disjointness	3732	10196	5022	19099	16287	31299
+ domain and range restrictions	3732	10196	5130	19233	16287	31299
+ all other $\mathcal{EL}^{++}$ axioms	7521	20449	14269	50218	33673	122221
every axiom[7]	7548	20478	15634	54452	47104	122221

Table 10.3: Number of classes, properties, and deterministic axioms in the LARGE BIOMED ontologies.

	NCI	SNOMED
FMA	3396	10760
NCI	-	18842

Table 10.4: Number of weighted equivalent axioms in the merged LARGE BIOMED alignments if no threshold is applied. The total number is 32998.

AROMA, GOMMA, LOGMAP, MAPSSS, SERVOMAP, and YAM++. For details about most of these systems and further references, we again refer to [AEE+12]. The unification of alignments and the mapping to a log-linear CBox $\mathcal{C}$ is performed exactly as described for the CONFERENCE benchmark in Section 10.3.1. The sizes of the unified alignments are mentioned in Table 10.4. The right part of Figurefig:datasets visualizes the structure of the LARGE BIOMED benchmark and, exemplarily, the merge process for the SNOMED and NCI ontology.

10.3.2 Ontology Learning

For the evaluation in ontology learning, we used the EKAW ontology from the CONFERENCE benchmark as the gold standard. It consists of 77 classes and 33 object properties. Furthermore, we have 71 subsumption, 74 disjointness, 39 domain and range, and 125 other $\mathcal{EL}^{++}$ axioms. When materialized, we observe 148 subsumption and 2,299 disjointness axioms. We refer to Table 10.1 for details.

We decided to generate confidence values using a "crowdsourcing" service. Probably the best known crowdsourcing platform is the *Amazon Mechanical Turk* (AMT)[14]. With AMT, Amazon offers numerous options for designing customized questionnaires. Due to its relatively high publicity (about 100,000 tasks were available at the time of this writing), it attracts a lot of users and consequently seems most suitable for our scenario.

We generated human intelligence tasks by creating questionnaires each with 10 yes/no questions. Half of these were used to generate confidence values for subsumption (disjointness) axioms. For the pair of class labels *Conference_Paper* and *Poster_Session*, for instance, the two types of yes/no questions were:

(a) Is every *Conference_Paper* also a *Poster_Session*?

(b) Can there be anything that is both a *Conference_Paper* and a *Poster_Session*?

For each pair of classes *and* for each type of question we obtained 9 responses from *different* AMT workers. A worker received 0.03 USD for each completed HIT. Overall, we paid 315.56 USD in worker compensation. The confidence value for a subsumption (disjointness) axiom was computed by dividing the number of "yes" ("no") answers by 9. We applied a threshold of 0.5, that is, only when the majority of the 9 workers answered with "yes" ("no") did we assign a confidence

[14] http://www.mturk.com

value to the axiom. Moreover, we halved the weights of the disjointness axioms for reasons of symmetry. This resulted in 2,507 axioms (84 subsumption and 2,423 disjointness) with confidence values.

10.4 Experimental Setup

As already mentioned in the introduction of this chapter, our experiments cover the two research questions Q13 and Q14. Question Q13 asks if the quality improves with increasing expressivity. In order to address this question, we perform experiments on ontologies with varying expressivity with our ELOG $\mathcal{EL}^{++}$ reasoner. The lowest level of expressiveness only contains weighted or deterministic subsumption axioms of form $A \sqsubseteq B$. In the next level of expressiveness, we add disjointness axioms like $A \sqcap B \sqsubseteq \bot$. Then, domain and range restrictions are added, and, finally, all remaining axioms entailed by the description logic $\mathcal{EL}^{++}$ are added.

Question Q14 states that computing the most probable coherent ontology with optimal algorithms, like our ELOG $\mathcal{EL}^{++}$ reasoner, lead to solutions with higher quality than the computation with approximate approaches. For verifying this claim, we conduct experiments with several approximate algorithms. In our ontology matching experiments, we compare our algorithm with the ontology alignment repair systems ALCOMO (see Section 10.2.2) and LOGMAP (refer to Section 10.2.2. Both systems are specialized for ontology matching and exploit the fact that weighted axioms are equivalent (subsumption) axioms and only occur *between* two ontologies.

In our ontology learning benchmark, we could not identify a system against which we could compare our algorithm. We refer to the related work section (Section 9.4) for in-depth discussions of potential systems and their differences. Thus, we compare our ELOG approach with a greedy algorithm which is often employed in ontology learning scenarios. The greedy algorithm sorts the axioms in descending order according to their confidence values and adds one axiom at a time to an initially empty ontology. However, it adds an axiom only if it does not render the resulting ontology incoherent. To compute precision and recall scores we materialized all subsumption and disjointness axioms of the resulting ontology. We used the reasoner Pellet for the materialization of axioms and the coherence checks.

In ontology matching and ontology learning, it is a common technique to apply a threshold α which means that only weighted axioms with a weight w with $w \geq \alpha$ are selected. The other axioms are dropped. The motivation behind this technique is the idea that weights below that threshold should not be considered because of too low quality. We conduct experiments with varying thresholds.

The CONFERENCE benchmark experiments and the experiments in ontology learning were performed on a virtual machine with 8 GB RAM and 2 cores with 2,4 Ghz. The LARGE BIOMED experiments were executed on a virtual machine with 60 GB RAM and 2 cores.

For measuring the quality, we applied different measures.

Objective As already used for evaluation of Part I in Section 6.3, we also compute the objective for most-probable coherent ontologies in log-linear description logics. The objective of a resulting CBox $\mathcal{C}'$ is defined as the non-normalized sum of the weights w_c of the weighted axioms c entailed in $\mathcal{O}$

$$Obj(\mathcal{C}') = \sum_{\{\langle c,w_c\rangle \in \mathcal{C}^{\mathsf{U}}:\mathcal{C}' \vDash c\}} w_c.$$

The objective is the straightest quality measure, since it is directly coupled with the result of the algorithm. If the algorithm is able to return a higher sum, the objective will be higher. We report the objective for every benchmark.

F-Measure Arguable, F-measure is the most widely used quality measure in information retrieval and was introduced by Rijsbergen in 1979 [vR79]. We summarize the definitions mainly from [MKSW99]. The main idea behind this measure is to compare the output of the system against a gold standard. This gold standard contains the correct outcome for this problem. The generation of such gold standards is often performed by human annotators. Gold standards are also often called reference and the systems output is often named hypothesis.

The f-measure returns a number between 0 and 1, where 0 means that the reference and the hypothesis are completely different and 1 means that they are equivalent. The f-measure is defined as the harmonic mean of precision and recall. Precision measures the percentage of how many elements in the reference were correctly found by the algorithm. Contrary, recall measures how many of the elements the algorithm has found were correct according to the reference. For their definition we thus introduce the total number in the reference N, the total number of elements in the hypothesis M, and the number of correct elements C. Correct elements are elements which occur in the reference and in the hypothesis. Now we are in the position to formally define precision P, recall R, and f-measure F as

$$P = \frac{C}{M}, \quad R = \frac{C}{N}, \quad \text{and} \quad F = \frac{2 \cdot P \cdot R}{P + R}.$$

Consequently, the f-measure returns the percentage how close the algorithm was able to compute the solution expected (and often generated) by humans. One goal of our experiments will be to experimentally verify that an increase in the objective also slightly increases the f-measure.

F-measure values are computed for the CONFERENCE and ontology learning benchmark. We did not provide f-measure results for the LARGE BIOMED benchmark because only a *silver* standard is provided. More reasons are given in Section 10.3.1.

Number of Unsatisfied Classes Counting the number of unsatisfied classes is a common measure in ontology debugging and was identified by Meilicke and Stuckenschmidt [MS08] as one of the key quality measures in ontology matching. As

the name implies, the measure returns the number of classes that are unsatisfiable. Formally, the measure is defined as

$$unsatC(\mathcal{C}') = |\{d|\mathcal{C}' \vDash d \sqsubseteq \bot\}|$$

where d represents a class within the CBox $\mathcal{C}'$. The lower the number of unsatisfied classes, the higher is the quality of the repaired ontology or the repaired alignment.

We compute the number of unsatisfied classes with the HERMIT [SMH08] reasoner since it is known from related publications [JRMGH13] that HERMIT outperforms other reasoners in the computation of unsatisfied classes. However, we were not able to compute the unsatisfied classes for the NCI and SNOMED pair under 5 hours, and thus are not able to provide the number of unsatisfied classes for the LARGE BIOMED benchmark.

10.5 Experimental Results

Our experimental result cover research questions Q13, which asks if increasing expressivity leads to higher quality, and Q14, which claims that optimal solving strategies like ELOG lead to higher quality compared to approximate approaches. In the ontology matching domain, we compare against specialized optimal and approximate alignment debugging algorithms. In ontology learning, we compare ELOG against a greedy approach. Question Q13 and Q14 are discussed in Section 10.5.1 and Section 10.5.2, respectively.

Furthermore, Section 10.5.3 contains results of ELOG on larger ontologies. We show that its runtime is faster and can cope with larger ontologies than other optimal approaches.

10.5.1 Increasing the Expressivity Improves the Quality

In order to measure if more expressivity also leads to higher quality we perform experiments with ontologies having different levels of expressivity. We conduct experiments in the areas of ontology matching and ontology learning. In case of ontology matching, we focus only on the CONFERENCE benchmark, since the LARGE BIOMED benchmark does not provide a gold-standard. For detailed explanations about the provided silver standard and why it is not sufficient for our purposes we refer the reader to 10.3.1.

Ontology Matching Benchmark Figure 10.4 visualizes the results for varying expressiveness and different thresholds for the CONFERENCE benchmark. Please note, that the scale in each chart changes at threshold 0.2 from 0.01 to 0.1 steps, since there exists none or only very few conflicts if applying high thresholds. Consequently, the differences between the expressiveness levels increase with lower thresholds in all charts. Hence, we focus on our following detailed analysis of

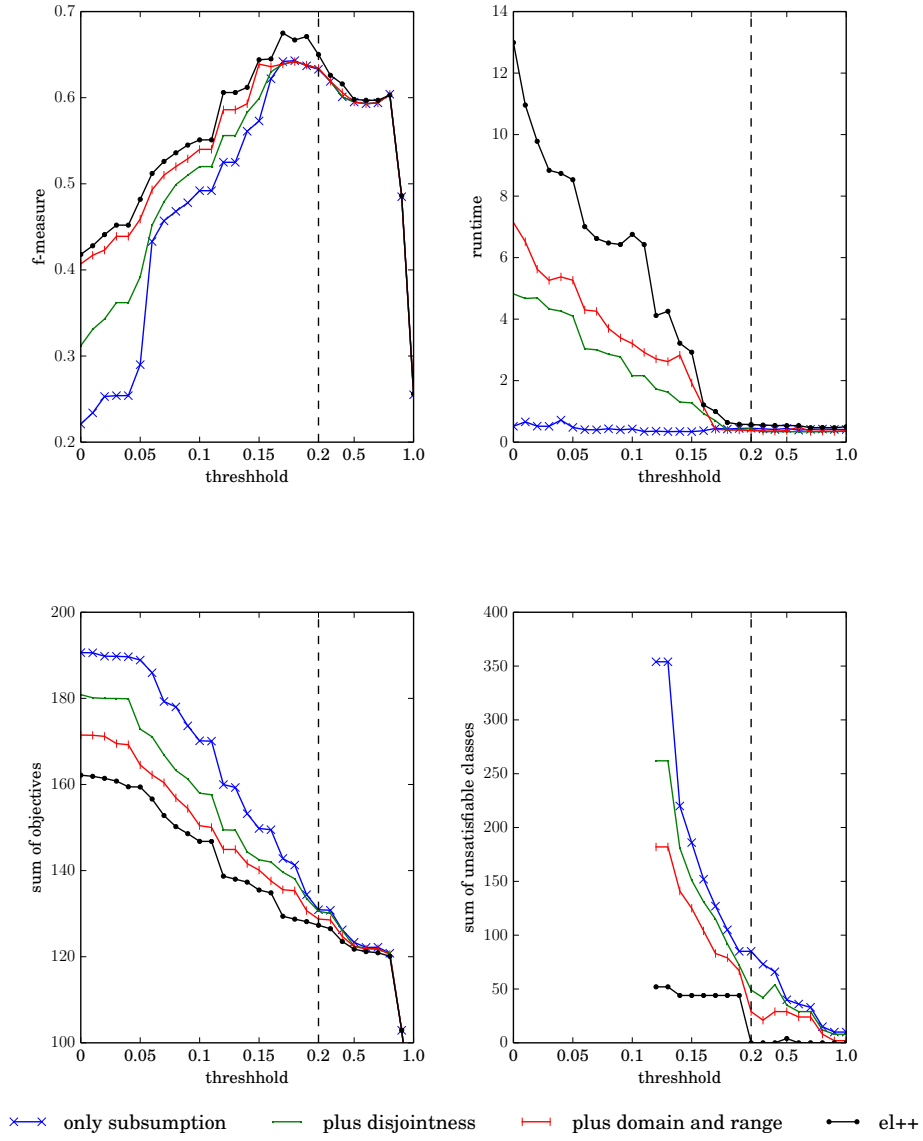

Figure 10.4: Results for different thresholds and increasing expressivity of the CONFERENCE benchmark. With increasing expressivity, f-measure and runtime (in seconds) are increasing. Contrary, the sum of the objectives and incoherent classes decrease. This effects are stronger the lower the threshold is since more conflicts occur. For thresholds lower than 0.12 the HERMIT reasoner failed in computing the number of incoherent classes. In total, the conference benchmark contains 2.973 classes.

the different quality measures on threshold areas below 0.2 since results for higher thresholds were almost equivalent for almost all quality measures.

With respect to f-measure (upper left figure) we observe a higher value with increasing expressivity. Thus, only evaluating subsumption axioms have lower f-measure scores than if disjointness axioms are added. Higher f-measures are achieved if domain and range axioms are also taken into account and the highest f-measure result is obtained if we incorporate full $\mathcal{EL}^{++}$ expressivity.

In terms of objective (lower left figure) we get the opposite picture. With increasing expressivity the objective decreases. The reason for this lies in the fact that with increasing expressivity more and more conflicts occur. Consequently, more and more correspondences are omitted due to incoherency. Since the number of weighted axioms stays constant, the objective decreases. Thus, in our setting the objective measures the amount of conflicts that have been avoided which is why we prefer low objectives over high objectives in our scenario. Please note, that this is the case because we apply the same optimal algorithm for every expressivity and every threshold.

The sum of unsatisfiable classes (lower right figure) is again higher with decreasing expressivity. In case we only include subsumption, we observe the highest number of unsatisfiable classes in the final alignment while we retrieve only very few unsatisfiable classes for full $\mathcal{EL}^{++}$ expressivity. Since low number of satisfiable classes is preferred over high numbers this shows an increase in quality. The reason why we obtain unsatisfied classes at all for $\mathcal{EL}^{++}$ expressivity is that the expressivity of our underlying ontologies is higher than $\mathcal{EL}^{++}$. However, as discussed in Section 8.1 many alternative more expressive DLs can be extended to log-linear description logics DLs which would result in lower numbers of unsatisfied classes or would even guarantee consistency.

Last, we provide the runtime in seconds in the lower right figure. We observe an increase in runtime with increasing number of resolved conflicts, since runtimes are higher for low thresholds. Furthermore, runtimes also increase with increasing expressivity.

In summary, our experiments on the CONFERENCE benchmark showed that the quality increases with increasing expressivity. Most importantly, f-measure results are higher. After our reinterpretation of the objective quality measure to a measure which returns the amount of avoided conflicts, we conclude that more conflicts are resolved with higher expressivity. Finally, the number of unsatisfiable classes is lower if expressivity increases which is again an increase in quality.

Ontology Learning Benchmark We performed experiments with varying expressivity on the ontology learning benchmark. The experimental results are given in Table 10.5. Since weighted axioms were given for subsumption and disjointness axioms, we decided to evaluate f-measure and objective results separately for subsumption and disjointness axioms. Furthermore, we include precision and recall in the evaluation.

Axiom type	Algorithm	Precision	Recall	F-Measure	Objective
Subsumption	only weighted axioms	0.609	0.611	0.610	59.9
	+ domain and range	0.709	0.599	0.649	56.0
	everything	0.840	0.568	0.677	54.2
Disjointness	only weighted axioms	0.948	0.960	0.954	1103
	+ domain and range	0.975	0.945	0.960	1056
	everything	0.992	0.937	0.964	1025

Table 10.5: Results for the ontology learning benchmark with increasing expressivity. Runtimes were 5.2 seconds with only weighted axioms and 5.6 seconds with everything.

If we examine the results for subsumption axioms, we observe an increase in Precision from 60.9% over 70.9% to 84% if we include only weighted subsumption and disjointness axioms, include domain and range axioms, and perform the experiments on full $\mathcal{EL}^{++}$, respectively. Precision increases if false positive axioms are removed due to conflict avoidance. Contrary to this, recall values decrease from 61.1% to 56.8% since some of the removed axioms are true positives. However, the increase in precision compensate the decrease in recall since we observe increasing f-measure values from 61% over 64.9% to 67.7%. Intuitively, we can conclude that more false positives are removed than true positives. Additionally, the objective also increases from 50.9% over 52% to 54.2% with increasing expressiveness.

Similar, but smaller effects are measured in case of disjointness axioms. For example, we only have an increase of f-measure of 1% between taking only weighted axioms and taking full $\mathcal{EL}^{++}$ expressivity. This is due to the very large number of true positive disjointness axioms. This large number evolves from the fact that we materialize disjointness axioms for evaluation. Since in this ontology, disjointness is often introduced on upper levels of the class hierarchy, we obtain many disjointness combinations in lower levels after materialization.

For both subsumption and disjointness axioms the objective decreases with increasing expressivity, which is a similar effect that we observed in the ontology matching benchmark above. With increasing expressivity more conflicts are resolved which lead to a lower objective. Runtimes for full $\mathcal{EL}^{++}$ axioms were slightly higher (5.6 seconds) than with only weighted axioms (5.2 seconds).

Conclusively, f-measure increased and the objective decreased with increasing expressivity in the ontology learning benchmark. Thus, we observed higher quality with increasing expressivity in both benchmarks.

10.5.2 ELOG has a Higher Quality Than Approximate Approaches

In this section, we experimentally address the question if optimal algorithms (like ELOG) lead to higher quality than approximate algorithms. To that end, we compare ELOG against different algorithms.

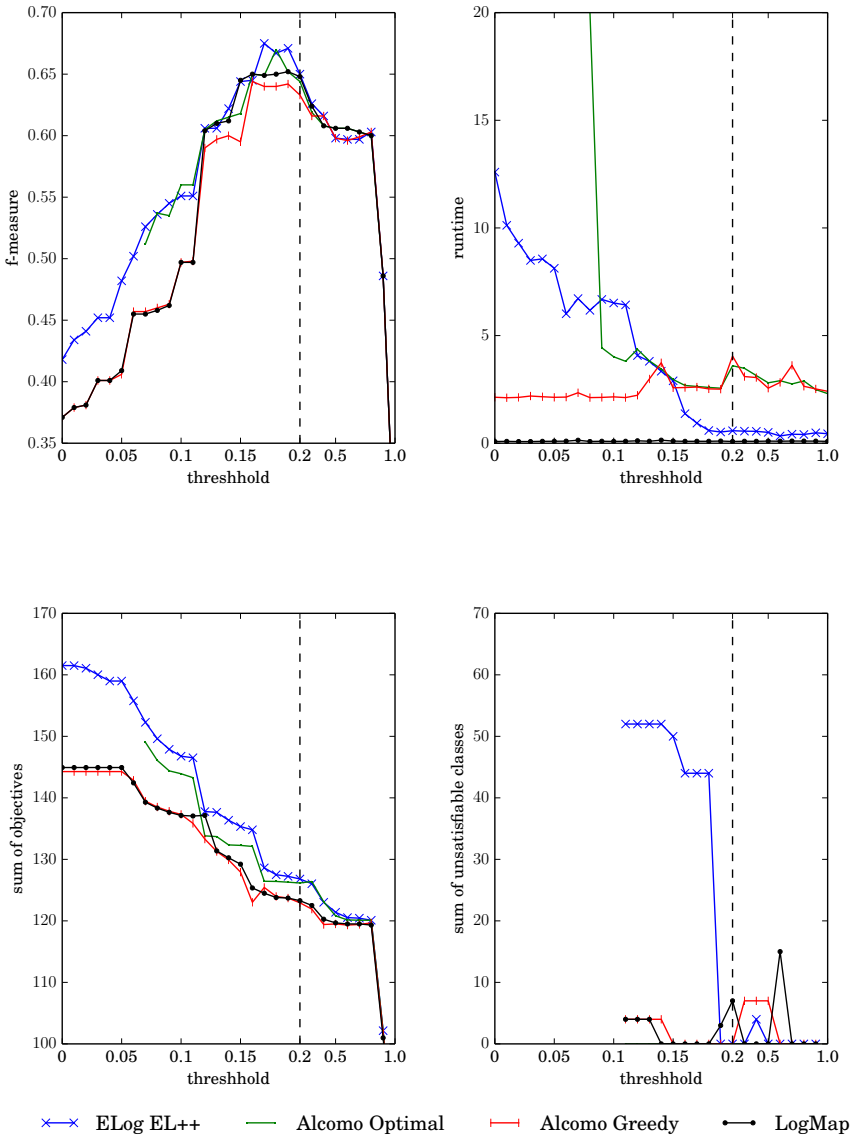

Figure 10.5: Results for ELOG compared with other approaches on the CONFERENCE benchmark. For thresholds lower than 0.12 the HERMIT reasoner failed in computing the number of incoherent classes. In total, the conference benchmark contains 2.973 classes.

Ontology Matching Benchmark In ontology matching we compare us against the alignment debugging tools LOGMAP and ALCOMO. The latter system provides a greedy and an optimal algorithm. For details about the systems and their algorithms, we refer to Section 10.2.2 and Section 10.2.2, respectively.

The results for the CONFERENCE benchmark are visualized in Figure 10.5. Again, we change the scale in each chart at threshold 0.2 from 0.01 to 0.1 steps, since there exists none or only very few conflicts if applying high thresholds and results were almost equivalent for all systems. In the following, we focus on the discussion of results for thresholds below 0.2.

Our system ELOG and the optimal algorithm of ALCOMO had the highest f-measure scores (upper left figure). The approximate algorithms of ALCOMO and LOGMAP reached lower f-measure scores. Similarly, the objective (lower left figure) of ELOG and the optimal algorithm of ALCOMO are higher than for the approximate algorithms of ALCOMO and LOGMAP.

ELOG has the highest number of unsatisfied classes (lower right figure) of all three algorithms. However, having 53 inconsistent classes is only 1.7% compared to the total sum of classes of 2.973. After some observations we discovered that one conflict often produces several inconsistent classes. Furthermore, all inconsistencies were caused from axioms which are out of the scope of $\mathcal{EL}^{++}$. The advantage of ELOG compared to the approximate algorithms of LOGMAP and ALCOMO is that it is proven to be coherent for the description logic $\mathcal{EL}^{++}$. Please also note, that many more expressive description logics than $\mathcal{EL}^{++}$ can be transformed to log-linear DLs. For an overview we refer to Section 8.1. Choosing a more expressive DL would lead to a reduction of the incoherent classes in our experiments. However, possible drawbacks like longer runtimes would have to be experimentally investigated. This is left to future work.

The attentive reader of the figures may be confused by the slightly higher objective results of ELOG compared to the optimal algorithm of ALCOMO. Since both algorithms are optimal, one might think that they must return similar results. However, ELOG has slightly more unsatisfied classes and thus detects slightly less conflicts. This leads to a slightly higher objective. The differences in f-measure originate from the fact that problems often have more than one optimal solution. Each of this optimal solution has the same objective but most probable different f-measure scores. Thus, ELog might choose a different one than the optimal algorithm of ALCOMO.

With respect to runtimes (upper right figure), the approximate algorithms of LOGMAP and ALCOMO were faster especially for lower thresholds. Our system ELOG, outperformed the approximate ALCOMO algorithm for thresholds higher than 0.15. Except for the range 0.11 and 0.12, the exact ALCOMO algorithm was slower than ELOG and did not terminate within one hour for thresholds below 0.09.

Overall, we can conclude that for the CONFERENCE benchmark our ELOG system is able to compete with state of the art alignment debugging tools. It achieves higher f-measure and objective scores than the approximate algorithms. With respect to runtime it outperforms the optimal algorithm of ALCOMO. This is remark-

Axiom type	Algorithm	Precision	Recall	F-Measure	Objective
Subsumption	Greedy	0.784	0.514	0.620	50.1
	ELOG	0.840	0.568	0.677	54.2
Disjointness	Greedy	0.935	0.990	0.961	1010
	ELOG	0.937	0.992	0.964	1025

Table 10.6: Results of ELOG compared with a greedy approach on the ontology learning benchmark. ELOG achieves a higher f-measure and objective. Runtimes of the greedy approach and ELOG were 40.3 and 5.6 seconds, respectively.

able since LOGMAP and ALCOMO are specialized on ontology matching. They leverage the fact that weighted axioms can only occur *between* ontologies and that those axioms are either subsumption or equivalence axioms. In ELOG it is however possible to attach weights to any arbitrary axiom.

Ontology Learning Benchmark Consequently, neither LOGMAP nor ALCOMO can deal with our ontology learning benchmark. Thus, we compare our ELOG system against a greedy algorithm which is described in Section 10.4.

As before, weighted subsumption and disjointness axioms are evaluated separately. Figure 10.6 summarizes the results. Precision, Recall, F-measure, and objective results were higher for the ELOG approach for subsumption axioms. Here, the greedy algorithm achieved 62% of f-measure while the ELOG algorithm reached 67.7% f-measure. The objective of the greedy algorithm is 50.1 compared to 54.2 in case of the optimal ELOG algorithm. For disjointness axioms we again observe a similar but weaker effect. Please refer to Section 10.5.1 for reasons why the effect is weaker.

With respect to runtimes, it is remarkable that ELOG's algorithm needs on average 5.6 seconds while the greedy algorithm requires 40.3 seconds. Thus, the optimal algorithm ELOG was over 7 times faster. Note that the 5.6 seconds include the time to classify the ontology. The reason for this huge time difference is that the greedy algorithm has to check the coherency of the ontology for every weighted axiom. Since the ontology learning benchmark contains a large number of weighted axioms, this procedure is slow.

In summary, these results indicate that ELOG is more effective and efficient than the greedy approach for small to medium sized ontologies.

10.5.3 ELOG can process large ontologies

In order to show the ability of ELOG to process larger ontologies, we perform experiment on the LARGE BIOMED ontology matching benchmark. Figure 10.6 visualizes runtime in seconds and Δ objective of ELOG, LOGMAP, and ALCOMO. The Δ objective is the result of the subtraction of the lowest objective $obj_{t,min}$ = $min(obj_{t,s} \forall s)$ from each objective $obj_{t,s}$ where t stands for a specific threshold

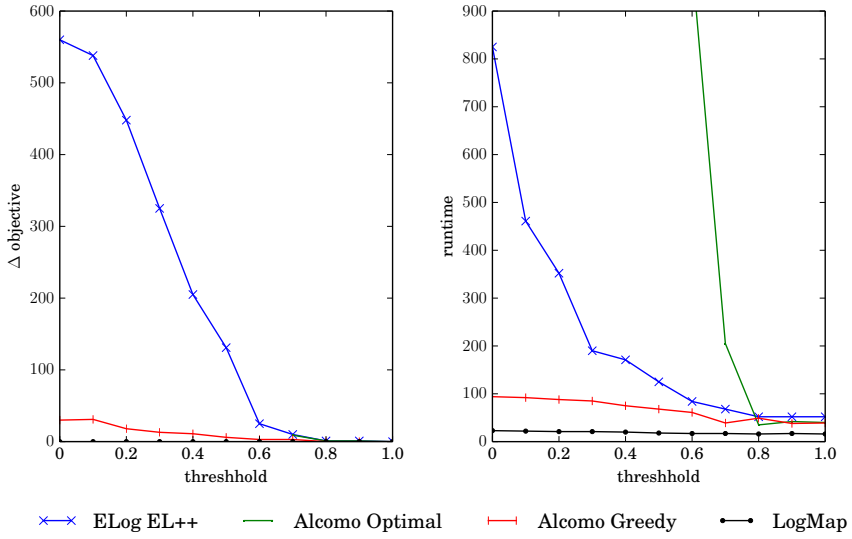

Figure 10.6: Results for ELOG compared with other approaches on the LARGE BIOMED benchmark.

and s for a system configuration. We decided to compute the Δ objective since otherwise the differences between the algorithms were not adequately visible.

The results verify that ELOG achieves higher objectives than the approximate algorithms of LOGMAP and ALCOMO where the objectives of LOGMAP are lower than the approximate ALCOMO algorithm. However, the approximate algorithms of LOGMAP and ALCOMO have lower runtimes and show a constant and linear behavior with lower thresholds, respectively. Contrary, the runtime of ROCKIT increases exponentially.

With respect to runtime, ELOG outperforms the optimal algorithm of ALCOMO which does not terminate within one hour for thresholds less than 0.7. This is in line with our runtime observation for the CONFERENCE benchmark (upper right chart of Figure 10.5) where the optimal algorithm of ALCOMO is also slower for most thresholds and is not able to compute a result for thresholds below 0.08.

Again, please note that this is remarkable, since ALCOMO is specialized for repairing ontology alignments only while ELOG can deal with any weighted axiom.

Chapter 11

Conclusion and Future Work

Finally, we conclude Part II of this thesis and draw future work perspectives. Furthermore, the link between Part I and Part II will be made explicit.

11.1 Conclusion

In Part II of this thesis we define log-linear description logics which allows to attach weights to arbitrary description logic axioms. Furthermore, a novel query type which computes the most-probable coherent ontology evolves from the semantics of log-linear description logics. Log-linear description logic were converted to a Markov logic network such that we can apply our techniques introduced in Part I for efficient MAP query solving. Thus, we applied or techniques in Part I to efficiently solve the most-probable coherent ontology of Part II. Our novel log-linear description logic reasoner ELOG implements log-linear description logics based on the DL $\mathcal{EL}^{++}$. However, our techniques can be applied to more expressive DLs as the ones mentioned in Section 8.1.

In the following, we answer the research questions from Section 1.2.2 in detail.

Q4 *How can we combine log-linear models with description logics and define the query of a most-probable coherent ontology?*

We showed that we can combine log-linear models with description logics by assigning weights to arbitrary description logic axioms. This splits our CBox into an uncertain CBox $\mathcal{C}^{\mathsf{U}}$ containing the weighted axioms and a certain CBox $\mathcal{C}^{\mathsf{D}}$ containing the non-weighted axioms. The latter one is assumed to be coherent, while $\mathcal{C}^{\mathsf{U}} \cup \mathcal{C}^{\mathsf{D}}$ usually contains conflicts. This describes the syntax of log-linear description logics in a nutshell. We refer to Section 8.2.1 for details.

The semantics assigns a probability to each deterministic CBox $\mathcal{C}'$. This probability is zero if CBox $\mathcal{C}'$ is incoherent or does not entail each axiom of the deterministic CBox $\mathcal{C}^{\mathsf{D}}$. This latter case ensures that each $\mathcal{C}'$ with a weight greater

than zero is a semantic *superset* of the given deterministic CBox $\mathcal{C}^{\mathsf{D}}$. If $\mathcal{C}'$ is coherent and entails $\mathcal{C}^{\mathsf{D}}$ the semantics follows a log-linear probability distribution. Section 8.2.2 formalizes these cases and gives descriptive examples.

The semantics of log-linear description logic allows to define a novel query type which asks for the most probable coherent ontology. The need for this novel query originates from the areas of ontology learning and ontology matching. We refer the reader to Section 10.2 for details. In both disciplines machine learning techniques are applied to create confidence values for description logic axioms. However, the resulting ontology is often incoherent. Thus, researchers aim to compute a coherent ontology while keeping as *many* information as possible.

Log-linear description logics define not only a possible semantic for the loosely coupled confidence values but also defines the maximum a-posteriori query which answers the following question:

"Given a log-linear $\mathcal{EL}^{++}$ CBox, what is a most probable coherent deterministic $\mathcal{EL}^{++}$ CBox over the same concept and role names?"

We refer the reader to Section 8.3.4 for formal definitions.

Q5 *Can we efficiently compute the most-probable coherent ontology utilizing the theory of Part I?*

The answer of Q5 is actually even more general. Section 8.3 shows how log-linear description logic CBoxes can be transformed to a Markov logic representation. With this Markov logic representation it is on the one hand possible to perform MAP inference and thus to compute the most probable coherent ontology. On the other hand, we can apply Markov logic techniques for marginal inference. This latter issue is, however, not addressed in this thesis. We refer to [NNS11] for details.

For transforming log-linear description logics into a Markov logic representation it is required that the underlying deterministic description logic has a finite set of materialization rules. Section 8.1 provides a list of such DLs and motivates our choice to demonstrate the procedure utilizing the description logic $\mathcal{EL}^{++}$.

The transformation requires four steps. First, the deterministic and uncertain CBox $\mathcal{C}^{\mathsf{D}}$ and $\mathcal{C}^{\mathsf{U}}$ is required to be normalized. When normalizing $\mathcal{C}^{\mathsf{U}}$ it is required to link each normalized axiom to its original axiom which can make normalization more challenging. The normalization is exemplarily performed for $\mathcal{EL}^{++}$ in Section 8.3.1.

In the second step we transform the set of materialization rules to first-order logic (see Section 8.3.2. Afterwards, we add the axioms from the deterministic CBox $\mathcal{C}^{\mathsf{D}}$ as evidence and the weighted axioms $\mathcal{C}^{\mathsf{U}}$ to the model. In the latter case, we have to make sure that the weight is correctly linked to the axiom. To that end, we iterate through all weighted non-normalized axioms, and add a weighted conjunction of all associated normalized axioms. This is the reason why it is required to link the normalized axioms and the non-normalized axioms. Section 8.3.3 approaches the addition of axioms.

Finally, Section 8.3.5 puts the components together and constructs an ML network. However, if we would like to apply the solving techniques of Part I, we require the ability to attach weights to conjunctions. In Markov logic we can only attach weights to clauses. If weights are attached to conjunctions, Richardson and Domingos [RD06] advise to split the conjunction of clauses into separate formulas, containing one clause each, and divide the weight by the number of clauses. This, however, leads to a different semantic. In order to be able to integrate conjunctions in a semantically correct way, we extend the semantic of Markov logic. We first apply existing methods [FdBR$^+$13, Jan04] to reduce a formula in conjunctive normal form to conjunctions of literals by introducing additional hard conjunctions. Then, we model conjunctions as ILP constraints, integrate them into CPI, and, finally, extend our CPA approach in a way that we can also aggregate conjunctions. Details are explained in Section 8.4.

Q6 *Can we experimentally verify that a solution's quality increase with increasing expressivity and that optimal solving strategies result in higher quality solutions than approximate solving strategies?*

For experimental verification, we performed exhaustive experiments which are described in Chapter 10. First, we implemented the translation of log-linear description logics to Markov logic in a new log-linear description logic solver ELOG which exploits ROCKIT from Part I for MAP query solving. Furthermore, we selected benchmarks from the main application areas ontology learning and ontology matching. As quality measures we introduced f-measure, objective, and number of incoherent classes.

With respect to the first part of Q6 we experimentally verified an increase in f-measure and a decrease in the number of incoherent classes with increasing expressivity for both the ontology learning and ontology matching benchmark. Furthermore, we observed a decrease in objective with increasing expressivity. This decrease is explained by the fact that more conflicts are resolved with increasing expressivity. In this scenario, it is required to reinterpret the objective as a measure of the amount of conflicts that have been avoided. In summary, we can conclude that the quality increases with increasing expressivity.

For addressing the second part of Q6, we compared ELOG with several approximate approaches. Our exhaustive literature review in Chapter 9 and in particular our analysis of presumably every existing probabilistic, possibilistic, or fuzzy description logic reasoner in Section 9.4 showed that none of these systems is able to compute a most-probable coherent ontology. Thus, we were only able to compare our system ELOG with the alignment debugging tools LOGMAP and ALCOMO. We refer to Section 10.2 for system descriptions. In ontology learning, we compared us against a greedy algorithm described in Section 10.4. In both the ontology learning and ontology matching benchmark ELOG achieved higher f-measure and objective scores than approximate algorithms. The number of unsatisfied classes were slightly higher, since the expressivity of the benchmarks were beyond $\mathcal{EL}^{++}$.

However, when implementing more expressive description logics (see Section 8.1 the number of unsatisfied classes would decrease. The advantage of log-linear description logics compared to the approximate approaches in ontology matching is that ontologies are guarantied to be coherent if they do not exceed the underlying DL (in our case $\mathcal{EL}^{++}$).

Additionally, we measured the runtime of the algorithms and performed experiments on larger ontologies (see Section 10.5.3). Generally, we observed an increase in runtime with increasing expressivity. Furthermore, we discovered that ELOG is able to process larger ontologies.

In the ontology learning experiments, runtimes of the greedy algorithm are significantly slower than our ELOG algorithm. In the ontology matching experiments, runtime of ELOG are (mostly) slower compared to the approximate algorithms of LOGMAP and ALCOMO. However, ELOG was faster than the optimal algorithm of ALCOMO in both the smaller and larger ontology matching benchmark. In fact, ALCOMO does not terminate for lower thresholds. This is remarkable since ALCOMO (and LOGMAP) are especially optimized for alignment repairing tasks. Thus, only weighted subsumption axioms between two ontologies are allowed. In ELOG it is possible to attach weights to every axiom type everywhere in the ontology.

11.2 Future Work

There are several possible future research directions. In the following, we point out the most promising ones.

Implement more expressive log-linear DLs Our implementation of log-linear description logics is based on the description logic $\mathcal{EL}^{++}$. However, there exist several more expressive description logics which can be transferred to log-linear description logics in a straight-forward way. Section 8.1 provides an incomplete list. In future, more implementations of different description logics are planned. One of the most promising extensions is $\mathcal{SROIQ} - RL$ since it is the underlying DL for the OWL 2 profile RL. In this context, it has to be experimentally determined if the increase in quality of a more expressive description logic like $\mathcal{SROIQ} - RL$ justifies the (probable) longer runtimes compared to $\mathcal{EL}^{++}$.

Set starting solutions In our experiments we noticed that especially in the field of ontology matching approximate algorithms generate comparable results in lower runtimes. These approximate solutions can be set as starting solutions in our solver ELOG. This starting solutions are then set as starting points in the ILP solvers for MAP state computation. Thus, this extension could also be seen as general extension for our MLN solver ROCKIT. However, in general Markov logic networks we experimentally prove that approximate algorithms like MAXWALKSAT produce less qualitative results in higher runtimes.

Integrate additional constraints One strength of our formulation of the most-probable coherent ontology query as Markov logic networks is the easy extendability with additional constraints. Thus, log-linear description logics are extendable with e.g. temporal restrictions. We could for instance add constraints such that a may not be a teacher of b if a died before b was born. An interesting research direction is the integration of fuzzy concepts into log-linear description logics by modeling the fuzziness with Markov logic constraints.

Reinterpret weights as utilities The semantics of log-linear description logics could presumably be changed such that weights can be interpreted as utilities [Fis70]. Then, solutions having higher utilities are preferred among those with lower utilities if these solutions are in conflict with each other. Ragone et al. [RND+09a, RND+09b], for example, model preferences as weighted formulas within a specific description logic. In their work, they assign utilities to axioms and maximize the utilities by maximizing their sum. In our ILP formulation of the MAP query, we also maximize the sum of the weights of the true description logic axioms. This connection has to be further elaborated and the semantics of log-linear description logics has to be adapted such that it follows the characteristics of utilities [Fis70].

Apply log-linear description logics Up to now, log-linear description logics have only been applied in activity recognition [HRN+12, HRS13]. In this thesis, we also performed experiments in the area of ontology learning and ontology matching. In future research, we plan to integrate log-linear description logics into existing ontology learning approaches. A promising starting point is the work of Fleischhacker et al. [FMVN13]. Additionally, we encourage other domains like the biomedical research area to add confidence values to the facts of their ontologies. In case of conflicts, these can then be resolved by computing the most probable coherent ontology.

Bibliography

[AB12] Udi Apsel and Ronen I. Brafman. Exploiting uniform assignments in first-order mpe. In Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012, pages 74–83. AUAI Press, 2012.

[ABB+00] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. Nature genetics, 25(1):25–29, 2000.

[AEE+12] José-Luis Aguirre, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Willem Robert van Hage, Laura Hollink, Christian Meilicke, Andriy Nikolov, Dominique Ritze, François Scharffe, Pavel Shvaiko, Ondrej Sváb-Zamazal, Cássia Trojahn dos Santos, Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Benjamin Zapilko. Results of the ontology alignment evaluation initiative 2012. In Proceedings of the 7th International Workshop on Ontology Matching, Boston, MA, USA, November 11, 2012, volume 946 of CEUR Workshop Proceedings. CEUR-WS.org, 2012.

[AKS11] Babak Ahmadi, Kristian Kersting, and Scott Sanner. Multi-evidence lifted message passing, with application to pagerank and the kalman filter. In IJCAI, pages 1152–1158, 2011.

[Apt99] Krzysztof R. Apt. The essence of constraint propagation. Theor. Comput. Sci., 221(1-2):179–210, 1999.

[ARMS03] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. IEEE Trans. on CAD of Integrated Circuits and Systems, 22(9):1117–1137, 2003.

[Asa06] Takao Asano. An improved analysis of goemans and williamson's lp-relaxation for max sat. Theor. Comput. Sci., 354(3):339–353, 2006.

[BB72] Umberto Bertele and Francesco Brioschi. Nonserial Dynamic Programming. Academic Press, Inc., Orlando, FL, USA, 1972.

[BBG88] Rochelle L. Boehning, Ralph M. Butler, and Billy E. Gillett. A parallel integer linear programming algorithm. European Journal of Operational Research, 34(3):393 – 398, 1988.

[BBL05a] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope. In IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005, pages 364–369. Professional Book Center, 2005.

[BBL05b] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope. ltcs-report ltcs-05-01, chair for automata theory. Institute for Theoretical Computer Science, Dresden University of Technology, Germany, page 37, 2005.

[BBL08] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope further. In In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions, 2008.

[BDGRS09] Fernando Bobillo, Miguel Delgado, Juan Gómez-Romero, and Umberto Straccia. Fuzzy description logics under gödel semantics. Int. J. Approx. Reasoning, 50(3):494–514, 2009.

[Bea09] Alan Beaulieu. Learning SQL - Master SQL Fundamentals (2. ed.). O'Reilly, 2009.

[BF98] Brian Borchers and Judith Furman. A two-phase exact algorithm for max-sat and weighted max-sat problems. Journal of Combinatorial Optimization, 2(4):299–306, 1998.

[BHD+11] James Blythe, Jerry R Hobbs, Pedro Domingos, Rohit J Kate, and Raymond J Mooney. Implementing weighted abduction in markov logic. In Proceedings of the Ninth International Conference on Computational Semantics, pages 55–64. Association for Computational Linguistics, 2011.

[BHR13] Hung Hai Bui, Tuyen N. Huynh, and Sebastian Riedel. Automorphism groups of graphical models and lifted variational inference. CoRR, abs/1309.6822, 2013.

[BK02] Christian Borgelt and Rudolf Kruse. Induction of association rules: Apriori implementation. In Wolfgang Härdle and Bernd Rönz, editors, Compstat, pages 395–400. Physica-Verlag HD, 2002.

[BLHL+01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. Scientific american, 284(5):28–37, 2001.

[BLRA11] Elena Bellodi, Evelina Lamma, Fabrizio Riguzzi, and Simone Albani. A distribution semantics for probabilistic ontologies. In Proceedings of the 7th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2011), Bonn, Germany, October 23, 2011, volume 778 of CEUR Workshop Proceedings, pages 75–86. CEUR-WS.org, 2011.

[BN07] Christopher M. Bishop and Nasser M. Nasrabadi. *Pattern Recognition and Machine Learning*. J. Electronic Imaging, 16(4):049901, 2007.

[BP11] Franz Baader and Rafael Peñaloza. Are fuzzy description logics with general concept inclusion axioms decidable? In FUZZ-IEEE 2011, IEEE International Conference on Fuzzy Systems, Taipei, Taiwan, 27-30 June, 2011, Proceedings, pages 1735–1742. IEEE, 2011.

[BP12] Stefan Borgwardt and Rafael Peñaloza. Undecidability of fuzzy description logics. In Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012. AAAI Press, 2012.

[Bra84] Kjell Bratbergsengen. Hashing methods and relational algebra operations. In Tenth International Conference on Very Large Data Bases, August 27-31, 1984, Singapore, Proceedings, pages 323–333. Morgan Kaufmann, 1984.

[Bre06] Christopher Brewster. *Ontology Learning from Text: Methods, Evaluation and Applications* paul buitelaar, philipp cimiano, and bernado magnini (editors) (dfki saarbrücken, university of karlsruhe, and itc-irst), amsterdam: Ios press (frontiers in artificial intelligence and applications, edited by j. breuker et al., volume 123), 2005, v+171 pp; hardbound, isbn 1-58603-523-1. Computational Linguistics, 32(4):569–572, 2006.

[BS13] Loris Bozzato and Luciano Serafini. Materialization calculus for contexts in the semantic web. In Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013, volume 1014 of CEUR Workshop Proceedings, pages 552–572. CEUR-WS.org, 2013.

[Cha98] Surajit Chaudhuri. An overview of query optimization in relational systems. In Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS '98, pages 34–43, New York, NY, USA, 1998. ACM.

[CL07] James Clarke and Mirella Lapata. Modelling compression with discourse constraints. In EMNLP-CoNLL 2007, Proceedings of the

2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic, pages 1–11. ACL, 2007.

[CLdC13] Rommel N. Carvalho, Kathryn B. Laskey, and Paulo Cesar G. da Costa. Pr-owl 2.0 - bridging the gap to owl semantics. In Uncertainty Reasoning for the Semantic Web II, International Workshops URSW 2008-2010 Held at ISWC and UniDL 2010 Held at FLoC, Revised Selected Papers, volume 7123 of Lecture Notes in Computer Science, pages 1–18. Springer, 2013.

[CM11] Robert Crane and Luke K McDowell. Evaluating markov logic networks for collective classification. In Proceedings of the 9th MLG Workshop at the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. San Diego, CA, USA, August, 20-21, 2011.

[CMSV09] Philipp Cimiano, Alexander Mädche, Steffen Staab, and Johanna Völker. Ontology learning. In Handbook on Ontologies, pages 245–267. Springer, 2009.

[CO93] Stephen J. Cannan and Gerard A. Otten. SQL - the standard handbook based on the new SQL standard: (ISO 9075:1992(E)). McGraw-Hill, 1993.

[Cos05] Paulo CG Costa. Bayesian semantics for the Semantic Web. George Mason University, 2005.

[CV05] Philipp Cimiano and Johanna Völker. Text2onto. In Natural Language Processing and Information Systems, 10th International Conference on Applications of Natural Language to Information Systems, NLDB 2005, Alicante, Spain, June 15-17, 2005, Proceedings, volume 3513 of Lecture Notes in Computer Science, pages 227–238. Springer, 2005.

[Dan51] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. New York, 1951.

[Dat89] Chris J. Date. A Guide to the SQL Standard, Second Edition. Addison-Wesley, 1989.

[dB11] Guy Van den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain, pages 1386–1394, 2011.

[DBK11] Torsten Dierkes, Martin Bichler, and Ramayya Krishnan. Estimating the effect of word of mouth on churn and cross-buying in the mobile phone market with markov logic networks. Decision Support Systems, 51(3):361–371, 2011.

[dBTM+11] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, pages 2178–2185. IJCAI/AAAI, 2011.

[dCL06] Paulo Cesar G. da Costa and Kathryn B. Laskey. Pr-owl: A framework for probabilistic ontologies. In Formal Ontology in Information Systems, Proceedings of the Fourth International Conference, FOIS 2006, Baltimore, Maryland, USA, November 9-11, 2006, volume 150 of Frontiers in Artificial Intelligence and Applications, pages 237–249. IOS Press, 2006.

[dCLC+08] Paulo Cesar G. da Costa, Marcelo Ladeira, Rommel N. Carvalho, Kathryn B. Laskey, Laécio L. Santos, and Shou Matsumoto. A first-order bayesian tool for probabilistic ontologies. In Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA, pages 631–636. AAAI Press, 2008.

[DD95] Hugh Darwen and C. J. Date. The third manifesto. SIGMOD Record, 24(1):39–49, 1995.

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. J. Log. Program., 1(3):267–284, 1984.

[Dia77] Persi Diaconis. Finite forms of de finetti's theorem on exchangeability. Synthese, 36(2):271–281, 1977.

[DJK+] Pedro Domingos, Dominik Jain, Stanley Kok, Daniel Lowd, Hoifung Poon, and Matthew Richardson. Alchemy website. http://alchemy.cs.washington.edu/. last visit: 19.05.2012.

[DLD+04] Robin Dhamankar, Yoonkyong Lee, Anhai Doan, Alon Halevy, and Pedro Domingos. imap: discovering complex semantic matches between database schemas. In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, ACM, pages 383–394. ACM, Press, 2004.

[DLK+08] Pedro Domingos, Daniel Lowd, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. Just add weights: Markov logic for the semantic web. In Uncertainty Reasoning for the Semantic Web I, ISWC International Workshops, URSW 2005-2007, Revised Selected and Invited Papers, volume 5327 of Lecture Notes in Computer Science, pages 1–25. Springer, 2008.

[DLP91] Didier Dubois, Jérôme Lang, and Henri Prade. A brief overview of possibilistic logic. In Symbolic and Quantitative Approaches to Reasoning and Uncertainty, European Conference, ECSQAU, Marseille, France, October 15-17, 1991, Proceedings, volume 548 of Lecture Notes in Computer Science, pages 53–57. Springer, 1991.

[DLP94] D Dubois, J Lang, and H Prade. Possibilistic logic. In Handbook of logic in artificial intelligence and logic programming (vol. 3), pages 439–513. Oxford University Press, Inc., 1994.

[DLSM04] Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for cnf. In Proceedings of the 41st annual Design Automation Conference, DAC '04, pages 530–534, New York, NY, USA, 2004. ACM.

[Dom07] Pedro Domingos. Toward knowledge-rich data mining. Data Mining and Knowledge Discovery, 15(1):21–28, 2007.

[DP04] Zhongli Ding and Yun Peng. A probabilistic extension to ontology language owl. In System Sciences, 2004. Proceedings of the 37th Annual Hawaii international conference on, pages 10–pp, 2004.

[dSBAR05] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005, pages 1319–1325. Professional Book Center, 2005.

[dSBAR06] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Mpe and partial inversion in lifted probabilistic variable elimination. In Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA, pages 1123–1130. AAAI Press, 2006.

[DSM08] Paul T. Darga, Karem A. Sakallah, and Igor L. Markov. Faster symmetry discovery using sparsity of symmetries. In Proceedings of the 45th Design Automation Conference, DAC 2008, Anaheim, CA, USA, June 8-13, 2008, pages 149–154. ACM, 2008.

[EIL+08] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. Artif. Intell., 172(12-13):1495–1539, 2008.

[EMS+11] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn. Ontology alignment evaluation initiative: Six years of experience. In Stefano Spaccapietra, editor, Journal on Data Semantics XV, volume 6720 of Lecture Notes in Computer Science, pages 158–192. Springer Berlin Heidelberg, 2011.

[ES07] Jérôme Euzenat and Pavel Shvaiko. Ontology matching. Springer, 2007.

[FdBR+13] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Sht. Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. CoRR, abs/1304.6810, 2013.

[FdBT+11] Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. Inference in probabilistic logic programs using weighted cnf's. In UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011, pages 211–220. AUAI Press, 2011.

[Fin72] B.D. Finetti. Probability, induction and statistics: the art of guessing. Probability and mathematical statistics. Wiley, 1972.

[Fis70] Peter C Fishburn. Utility theory for decision making. Technical report, DTIC Document, 1970.

[FL07] Ildikó Flesch and Peter J.F. Lucas. Markov equivalence in bayesian networks. In Peter Lucas, José A. Gámez, and Antonio Salmerón, editors, Advances in Probabilistic Graphical Models, volume 214 of Studies in Fuzziness and Soft Computing, pages 3–38. Springer Berlin Heidelberg, 2007.

[FMVN13] Daniel Fleischhacker, Christian Meilicke, Johanna Völker, and Mathias Niepert. Computing incoherence explanations for learned ontologies. In Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, Lecture Notes in Computer Science, pages 80–94. Springer, 2013.

[GD10] Vibhav Gogate and Pedro Domingos. Exploiting logical structure in lifted probabilistic inference. In Statistical Relational Artificial Intelligence, Papers from the 2010 AAAI Workshop,

Atlanta, Georgia, USA, July 12, 2010, volume WS-10-06 of AAAI Workshops. AAAI, 2010.

[GD12] Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. CoRR, abs/1202.3724, 2012.

[GK07] Bernd Gärtner and Volker Kaibel. Two new bounds for the random-edge simplex-algorithm. SIAM J. Discrete Math., 21(1):178–190, 2007.

[GL02] Rosalba Giugno and Thomas Lukasiewicz. P- $\mathcal{SHOQ}$ (d): A probabilistic extension of $\mathcal{SHOQ}$ (d) for probabilistic ontologies in the semantic web. In Sergio Flesca, Sergio Greco, Giovambattista Ianni, and Nicola Leone, editors, Logics in Artificial Intelligence, volume 2424 of Lecture Notes in Computer Science, pages 86–97. Springer Berlin Heidelberg, 2002.

[GN88] Michael R. Genesereth and Nils J. Nilsson. Logical foundations of artificial intelligence. Morgan Kaufmann, 1988.

[GRS96] WR Gilks, S Richardson, and DJ Spiegelhalter. Markov chain monte carlo in practice. Interdisciplinary statistics, 1996.

[Häh01] Reiner Hähnle. Advanced many-valued logics. In D.M. Gabbay and F. Guenthner, editors, Handbook of Philosophical Logic, volume 2 of Handbook of Philosophical Logic, pages 297–395. Springer Netherlands, 2001.

[Hai96] Theodore Hailperin. Sentential probability logic: Origins, development, current status, and technical applications. Lehigh University Press, 1996.

[Háj01] Alan Hájek. Probability, logic, and probability logic. The Blackwell guide to philosophical logic, pages 362–384, 2001.

[HBLL10] Eun Young Ha, Alok Baikadi, Carlyle Licata, and James C Lester. Ncsu: Modeling temporal relations with markov logic and lexical ontology. In Proceedings of the 5th International Workshop on Semantic Evaluation, pages 341–344. Association for Computational Linguistics, 2010.

[Hei94] Jochen Heinsohn. Probabilistic description logics. In UAI '94: Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence, Seattle, Washington, USA, July 29-31, 1994, pages 311–318. Morgan Kaufmann, 1994.

[HLH90] F.S. Hillier, G.J. Lieberman, and M. Hillier. Introduction to operations research, volume 6. McGraw-Hill New York, NY, 1990.

[HM09] Tuyen N. Huynh and Raymond J. Mooney. Max-margin weight learning for markov logic networks. In Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part I, volume 5781 of Lecture Notes in Computer Science, pages 564–579. Springer, 2009.

[HMS10] Hans-Jörg Happel, Walid Maalej, and Stefan Seedorf. Applications of ontologies in collaborative software development. In Collaborative Software Engineering, pages 109–129. Springer, 2010.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Systems Science and Cybernetics, 4(2):100–107, 1968.

[HNS11] Rim Helaoui, Mathias Niepert, and Heiner Stuckenschmidt. Recognizing interleaved and concurrent activities: A statistical-relational approach. In Ninth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2011, 21-25 March 2011, Seattle, WA, USA, Proceedings, pages 1–9. IEEE, 2011.

[Hol13] Bernhard Hollunder. An alternative proof method for possibilistic logic and its application to terminological logics. CoRR, abs/1302.6819, 2013.

[HRN+12] Rim Helaoui, Daniele Riboni, Mathias Niepert, Claudio Bettini, and Heiner Stuckenschmidt. Towards activity recognition using probabilistic description logics. Activity Context Representation: Techniques and Languages, AAAI Technical Report WS-12-05, pages 26–31, 2012.

[HRS13] Rim Helaoui, Daniele Riboni, and Heiner Stuckenschmidt. A probabilistic ontological framework for the recognition of multilevel human activities. In The 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '13, Zurich, Switzerland, September 8-12, 2013, pages 345–354. ACM, 2013.

[HSNM11] Jakob Huber, Timo Sztyler, Jan Nößner, and Christian Meilicke. Codi: Combinatorial optimization for data integration: results for oaei 2011. In Proceedings of the 6th International Workshop on Ontology Matching, Bonn, Germany, October 24, 2011, volume 814 of CEUR Workshop Proceedings. CEUR-WS.org, 2011.

[HST00] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. Logic Journal of the IGPL, 8(3):239–263, 2000.

[Isi25] Ernst Ising. Beitrag zur theorie des ferromagnetismus. Zeitschrift für Physik, 31(1):253–258, 1925.

[Jae94] Manfred Jaeger. Probabilistic reasoning in terminological logics. In Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94). Bonn, Germany, May 24-27, 1994, pages 305–316. Morgan Kaufmann, 1994.

[Jan04] Tomi Janhunen. Representing normal programs with clauses. In Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004, pages 358–362. IOS Press, 2004.

[JGMS10] Abhay Kumar Jha, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. Lifted inference seen from the other side : The tractable features. In Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada, pages 973–981. Curran Associates, Inc., 2010.

[JRG11] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I, volume 7031 of Lecture Notes in Computer Science, pages 273–288. Springer, 2011.

[JRMGH13] Ernesto Jiménez-Ruiz, Christian Meilicke, Bernardo Cuenca Grau, and Ian Horrocks. Evaluating mapping repair systems with large biomedical ontologies. In Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013, volume 1014 of CEUR Workshop Proceedings, pages 246–257. CEUR-WS.org, 2013.

[KAA+11] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans D. Mittelmann, Ted K. Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. Miplib 2010. Math. Program. Comput., 3(2):103–163, 2011.

[KAN09] Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009, pages 277–284. AUAI Press, 2009.

[Kaz09] Yevgeny Kazakov. Consequence-driven reasoning for horn shiq ontologies. In IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009, pages 2040–2045, 2009.

[KD08] Stanley Kok and Pedro Domingos. Extracting semantic networks from text via relational clustering. In Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I, volume 5211 of Lecture Notes in Computer Science, pages 624–639. Springer, 2008.

[KF09] Daphne Koller and Nir Friedman. Probabilistic Graphical Models - Principles and Techniques. MIT Press, 2009.

[KIBFT11] Gabriele Kern-Isberner, Christoph Beierle, Marc Finthammer, and Matthias Thimm. Probabilistic logics in expert systems: Approaches, implementations, and applications. In Database and Expert Systems Applications - 22nd International Conference, DEXA 2011, Toulouse, France, August 29 - September 2, 2011. Proceedings, Part I, volume 6860 of Lecture Notes in Computer Science, pages 27–46. Springer, 2011.

[Kli11] Pavel Klinov. Practical reasoning in probabilistic description logic. The University of Manchester, Manchester, UK, 2011.

[KLP97] Daphne Koller, Alon Y. Levy, and Avi Pfeffer. P-classic: A tractable probablistic description logic. In Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, pages 390–397. AAAI Press / The MIT Press, 1997.

[Kok10] Stanley Kok. Structure Learning in Markov Logic Networks. PhD thesis, University of Washington, 2010.

[Kol06] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. IEEE Trans. Pattern Anal. Mach. Intell., 28(10):1568–1583, 2006.

[KP09] Jacek Kisynski and David Poole. Lifted aggregation in directed first-order probabilistic models. In IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009, pages 1922–1929, 2009.

[KPRY05] P. Koomen, V. Punyakanok, D. Roth, and W. Yih. Generalized inference with multiple semantic role labeling systems shared task pa-

per. In Ido Dagan and Dan Gildea, editors, CoNLL, pages 181–184, 2005.

[Krö10] Markus Krötzsch. Efficient inferencing for owl el. In Logics in Artificial Intelligence - 12th European Conference, JELIA 2010, Helsinki, Finland, September 13-15, 2010. Proceedings, volume 6341 of Lecture Notes in Computer Science, pages 234–246. Springer, 2010.

[KS80] R. Kindermann and J.L. Snell. Markov random fields and their applications, volume 1. Amer Mathematical Society, 1980.

[KSJ97] Henry Kautz, Bart Selman, and Yueyen Jiang. A general stochastic approach to solving problems with hard and soft constraints. The Satisfiability Problem: Theory and Applications, 17:573–586, 1997.

[Las08] Kathryn B. Laskey. Mebn: A language for first-order bayesian knowledge bases. Artif. Intell., 172(2-3):140–178, 2008.

[LD07] Daniel Lowd and Pedro Domingos. Efficient weight learning for markov logic networks. In Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Warsaw, Poland, September 17-21, 2007, Proceedings, volume 4702 of Lecture Notes in Computer Science, pages 200–211. Springer, 2007.

[LdC12] Kathryn Blackmond Laskey and Paulo Cesar G. da Costa. Of starships and klingons: Bayesian logic for the 23rd century. CoRR, abs/1207.1354, 2012.

[LF09] Marco Lippi and Paolo Frasconi. Prediction of protein beta-residue contacts by markov logic networks with grounding-specific weights. Bioinformatics, 25(18):2326–2333, 2009.

[LGK+10] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. In Proceedings of UAI, pages 340–349, 2010.

[LLR81] LD Landau, EM Lifshitz, and Linda E Reichl. Statistical physics, part 1. Physics Today, 34:74, 1981.

[LLS06] Anita C. Liang, Boris Lauser, and Margherita Sini. From agrovoc to the agricultural ontology service / concept server - an owl model for creating ontologies in the agricultural domain. In Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11, 2006, volume 216 of CEUR Workshop Proceedings. CEUR-WS.org, 2006.

[LS08] Thomas Lukasiewicz and Umberto Straccia. Managing uncertainty and vagueness in description logics for the semantic web. J. Web Sem., 6(4):291–308, 2008.

[Luk01a] Thomas Lukasiewicz. Probabilistic logic programming under inheritance with overriding. In UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001, pages 329–336. Morgan Kaufmann, 2001.

[Luk01b] Thomas Lukasiewicz. Probabilistic logic programming with conditional constraints. ACM Trans. Comput. Log., 2(3):289–339, 2001.

[Luk07] Thomas Lukasiewicz. Probabilistic description logic programs. Int. J. Approx. Reasoning, 45(2):288–307, 2007.

[Luk08] Thomas Lukasiewicz. Expressive probabilistic description logics. Artif. Intell., 172(6-7):852–883, 2008.

[LW66] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. Operations research, 14(4):699–719, 1966.

[Mac77] Alan K. Mackworth. Consistency in networks of relations. Artif. Intell., 8(1):99–118, 1977.

[MAK12] Martin Mladenov, Babak Ahmadi, and Kristian Kersting. Lifted linear programming. In Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, April 21-23, 2012, volume 22 of JMLR Proceedings, pages 788–797. JMLR.org, 2012.

[Mar03] François Margot. Exploiting orbits in symmetric ilp. Math. Program., 98(1-3):3–21, 2003.

[Mar10] Franois Margot. Symmetry in integer linear programming. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, 50 Years of Integer Programming 1958-2008, pages 647–686. Springer Berlin Heidelberg, 2010.

[Mei11] Christian Meilicke. Alignment incoherence in ontology matching. PhD thesis, PhD thesis, University Mannheim, 2011.

[MGH+09] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. Owl 2 web ontology language: Profiles. W3C recommendation, 27:61, 2009.

[MH09] Ralf Möller and Volker Haarslev. Tableau-based reasoning. In Handbook on Ontologies, pages 509–528. Springer, 2009.

[Min72] George J Minty. How good is the simplex algorithm? victor klee* department of mathematics, university of washington, seattle, washington and. Inequalities-III: proceedings, page 159, 1972.

[MKSW99] John Makhoul, Francis Kubala, Richard Schwartz, and Ralph Weischedel. Performance measures for information extraction. In In Proceedings of DARPA Broadcast News Workshop, pages 249–252, 1999.

[MNJ05] Prasenjit Mitra, Natasha F. Noy, and Anuj R. Jaiswal. Omen: A probabilistic ontology mapping tool. In The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings, volume 3729 of Lecture Notes in Computer Science, pages 537–547. Springer, 2005.

[MNRS00] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. Inf. Retr., 3(2):127–163, 2000.

[MPSP+09] Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. Owl 2 web ontology language: Structural specification and functional-style syntax. W3C recommendation, 27:17, 2009.

[MRR09a] Ivan Meza-Ruiz and Sebastian Riedel. Jointly identifying predicates, arguments and senses using markov logic. In Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 155–163. Association for Computational Linguistics, 2009.

[MRR09b] Ivan Meza-Ruiz and Sebastian Riedel. Multilingual semantic role labelling with markov logic. In Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task, pages 85–90. Association for Computational Linguistics, 2009.

[MRRL08a] I. Meza-Ruiz, S. Riedel, and O. Lemon. Spoken language understanding in dialogue systems, using a 2-layer markov logic network: improving semantic accuracy. In Proceedings of the 12th Workshop on the Semantics and Pragmatics of Dialogue (LONDIAL08), 2008.

[MRRL08b] Iván V. Meza-Ruiz, Sebastian Riedel, and Oliver Lemon. Accurate statistical spoken language understanding from limited development resources. In Proceedings of the IEEE International Conference on

Acoustics, Speech, and Signal Processing, ICASSP 2008, March 30 - April 4, 2008, Caesars Palace, Las Vegas, Nevada, USA, pages 5021–5024. IEEE, 2008.

[MS04] Alexander Maedche and Steffen Staab. Ontology learning. In Steffen Staab and Rudi Studer, editors, Handbook on Ontologies, International Handbooks on Information Systems, pages 173–190. Springer Berlin Heidelberg, 2004.

[MS08] Christian Meilicke and Heiner Stuckenschmidt. Incoherence as a basis for measuring the quality of ontology mappings. In Proceedings of the 3rd International Workshop on Ontology Matching (OM-2008) Collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26, 2008, volume 431 of CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings, volume 4603 of Lecture Notes in Computer Science, pages 67–83. Springer, 2007.

[MST07] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Repairing ontology mappings. In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, pages 1408–1413. AAAI Press, 2007.

[MVH+04] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. W3C recommendation, 10(2004-03):10, 2004.

[MZK+08] Brian Milch, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted probabilistic inference with counting formulas. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, pages 1062–1068. AAAI Press, 2008.

[NB03] Daniele Nardi and Ronald J. Brachman. An introduction to description logics. In The Description Logic Handbook: Theory, Implementation, and Applications, pages 1–40. Cambridge University Press, 2003.

[Néd05] C Nédellec. Learning language in logic-genic interaction extraction challenge. Contributors to the Challenge task, page 31, 2005.

[Nie12] Mathias Niepert. Markov chains on orbits of permutation groups. In Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012, pages 624–633. AUAI Press, 2012.

[Nie13] Mathias Niepert. Symmetry-aware marginal density estimation. In Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA. AAAI Press, 2013.

[Nil86] Nils J. Nilsson. Probabilistic logic. Artif. Intell., 28(1):71–87, 1986.

[NMS10] Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. A probabilistic-logical framework for ontology matching. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010. AAAI Press, 2010.

[NN10] Jan Noessner and Mathias Niepert. Codi: Combinatorial optimization for data integration: results for oaei 2010. In Proceedings of the 5th International Workshop on Ontology Matching (OM-2010), Shanghai, China, November 7, 2010, volume 689 of CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[NN11] Jan Noessner and Mathias Niepert. Elog: A probabilistic reasoner for owl el. In Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Proceedings, volume 6902 of Lecture Notes in Computer Science, pages 281–286. Springer, 2011.

[NNMS10] Jan Noessner, Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. Leveraging terminological structure for object reconciliation. In The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II, volume 6089 of Lecture Notes in Computer Science, pages 334–348. Springer, 2010.

[NNS11] Mathias Niepert, Jan Noessner, and Heiner Stuckenschmidt. Log-linear description logics. In IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, pages 2153–2158. IJCAI/AAAI, 2011.

[NNS13] Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In Proceedings of the Twenty-Seventh AAAI

Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA. AAAI Press, 2013.

[Nov06] Vilém Novák. Which logic is the real fuzzy logic? Fuzzy Sets and Systems, 157(5):635–641, 2006.

[NRDS11] Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. PVLDB, 4(6):373–384, 2011.

[OLRS07] James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio. Orbital branching. In Matteo Fischetti and DavidP. Williamson, editors, Integer Programming and Combinatorial Optimization, volume 4513 of Lecture Notes in Computer Science, pages 104–118. Springer Berlin Heidelberg, 2007.

[OLRS11] James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio. Orbital branching. Math. Program., 126(1):147–178, 2011.

[ORS10] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Worst-case optimal reasoning for the horn-dl fragments of owl 1 and 2. In Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010. AAAI Press, 2010.

[PD07] Hoifung Poon and Pedro Domingos. Joint inference in information extraction. In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, pages 913–918. AAAI Press, 2007.

[PD08] Hoifung Poon and Pedro Domingos. Joint unsupervised coreference resolution with markov logic. In 2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 650–659. ACL, 2008.

[Pea89] Judea Pearl. Probabilistic reasoning in intelligent systems - networks of plausible inference. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[Poo03] David Poole. First-order probabilistic inference. In IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003, pages 985–991. Morgan Kaufmann, 2003.

[Poo08] David Poole. The independent choice logic and beyond. In Probabilistic Inductive Logic Programming - Theory and Applications, volume 4911 of Lecture Notes in Computer Science, pages 222–243. Springer, 2008.

[Pre74] C.J. Preston. Gibbs states on countable sets, volume 68. Cambridge University Press Cambridge, 1974.

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging owl ontologies. In Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005, pages 633–640. ACM, 2005.

[Pug06] Jean-Franois Puget. Symmetry breaking revisited. In Pascal Hentenryck, editor, Principles and Practice of Constraint Programming - CP 2002, volume 2470 of Lecture Notes in Computer Science, pages 446–461. Springer Berlin Heidelberg, 2006.

[QJPD10] Guilin Qi, Qiu Ji, Jeff Z. Pan, and Jianfeng Du. Possdl - a possibilistic dl reasoner for uncertainty reasoning and inconsistency handling. In The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II, volume 6089 of Lecture Notes in Computer Science, pages 416–420. Springer, 2010.

[QPJ07] Guilin Qi, Jeff Z. Pan, and Qiu Ji. Extending description logics with uncertainty reasoning in possibilistic logic. In Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 9th European Conference, ECSQARU 2007, Hammamet, Tunisia, October 31 - November 2, 2007, Proceedings, volume 4724 of Lecture Notes in Computer Science, pages 828–839. Springer, 2007.

[RBLZ12] Fabrizio Riguzzi, Elena Bellodi, Evelina Lamma, and Riccardo Zese. Epistemic and statistical probabilistic ontologies. In Proceedings of the 8th International Workshop on Uncertainty Reasoning for the Semantic Web, Boston, USA, November 11, 2012, volume 900 of CEUR Workshop Proceedings, pages 3–14. CEUR-WS.org, 2012.

[RBLZ13] Fabrizio Riguzzi, Elena Bellodi, Evelina Lamma, and Riccardo Zese. Bundle: A reasoner for probabilistic ontologies. In Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, volume 7994 of Lecture Notes in Computer Science, pages 183–197. Springer, 2013.

[RCTT09] Sebastian Riedel, Hong-Woo Chun, Toshihisa Takagi, and Jun'ichi Tsujii. A markov logic approach to bio-molecular event extraction.

In Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing: Shared Task, pages 41–49. Association for Computational Linguistics, 2009.

[RD06] Matthew Richardson and Pedro Domingos. Markov logic networks. Machine Learning, 62(1-2):107–136, 2006.

[Rie08] Sebastian Riedel. Improving the accuracy and efficiency of map inference for markov logic. In UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008, pages 468–475. AUAI Press, 2008.

[Rie09] Sebastian Riedel. Efficient prediction of relational structure and its application to natural language processing. The University of Edinburgh, 2009.

[RK05] Sebastian Riedel and Ewan Klein. Genic interaction extraction with semantic and syntactic chains. In In Proceedings of the Fourth Workshop on Learning Language in Logic, pages 69–74, 2005.

[RKT07] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007, pages 2462–2467, 2007.

[RMR08] Sebastian Riedel and Ivan Meza-Ruiz. Collective semantic role labelling with markov logic. In Proceedings of the Twelfth Conference on Computational Natural Language Learning, CoNLL '08, pages 193–197, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[RMSZS09] Dominique Ritze, Christian Meilicke, Ondrej Sváb-Zamazal, and Heiner Stuckenschmidt. A pattern-based ontology matching approach for detecting complex correspondences. In Proceedings of the 4th International Workshop on Ontology Matching (OM-2009) collocated with the 8th International Semantic Web Conference (ISWC-2009) Chantilly, USA, October 25, 2009, volume 551 of CEUR Workshop Proceedings. CEUR-WS.org, 2009.

[RND+09a] Azzurra Ragone, Tommaso Di Noia, Francesco M. Donini, Eugenio Di Sciascio, and Michael P. Wellman. Computing utility from weighted description logic preference formulas. In Declarative Agent Languages and Technologies VII, 7th International Workshop, DALT 2009, Budapest, Hungary, May 11, 2009. Revised Selected and Invited Papers, volume 5948 of Lecture Notes in Computer Science, pages 158–173. Springer, 2009.

[RND+09b] Azzurra Ragone, Tommaso Di Noia, Francesco M. Donini, Eugenio Di Sciascio, and Michael P. Wellman. Weighted description logics preference formulas for multiattribute negotiation. In Scalable Uncertainty Management, Third International Conference, SUM 2009, Washington, DC, USA, September 28-30, 2009. Proceedings, volume 5785 of Lecture Notes in Computer Science, pages 193–205. Springer, 2009.

[Rot96] Dan Roth. On the hardness of approximate reasoning. Artif. Intell., 82(1-2):273–302, 1996.

[RRR06] A Ravindran, Gintaras Victor Reklaitis, and Kenneth Martin Ragsdell. Engineering optimization: Methods and applications. John Wiley & Sons, 2006.

[RSFF12] Patrick Rodler, Kostyantyn M. Shchekotykhin, Philipp Fleiss, and Gerhard Friedrich. Rio: minimizing user interaction in debugging of aligned ontologies. In Proceedings of the 7th International Workshop on Ontology Matching, Boston, MA, USA, November 11, 2012, volume 946 of CEUR Workshop Proceedings. CEUR-WS.org, 2012.

[RVMSZ10] Dominique Ritze, Johanna Völker, Christian Meilicke, and Ondrej Sváb-Zamazal. Linguistic analysis for complex ontology matching. In Proceedings of the 5th International Workshop on Ontology Matching (OM-2010), Shanghai, China, November 7, 2010, volume 689 of CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[SB07] Umberto Straccia and Fernando Bobillo. Mixed integer programming, general concept inclusions and fuzzy description logics. In New Dimensions in Fuzzy Logic and Related Technologies. Proceedings of the 5th EUSFLAT Conference, Ostrava, Czech Republic, September 11-14, 2007, Volume 2: Regular Sessions, pages 213–220. Universitas Ostraviensis, 2007.

[SBS+11] Sandeepkumar Satpal, Sahely Bhadra, Sundararajan Sellamanickam, Rajeev Rastogi, and Prithviraj Sen. Web information extraction using markov logic networks. In Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011 (Companion Volume), pages 115–116. ACM, 2011.

[Sch99] Alexander Schrijver. Theory of linear and integer programming. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[SD06a] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In Proceedings of the 6th IEEE International Conference on

Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China, pages 572–582. IEEE Computer Society, 2006.

[SD06b] Parag Singla and Pedro Domingos. Memory-efficient inference in relational domains. In Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA, pages 488–493. AAAI Press, 2006.

[SD08] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, pages 1094–1099. AAAI Press, 2008.

[SDG09] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Bisimulation-based approximate lifted inference. In UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009, pages 496–505. AUAI Press, 2009.

[SE13] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: State of the art and future challenges. IEEE Trans. Knowl. Data Eng., 25(1):158–176, 2013.

[SFFR12] Kostyantyn M. Shchekotykhin, Gerhard Friedrich, Philipp Fleiss, and Patrick Rodler. Interactive ontology debugging: Two query strategies for efficient fault localization. J. Web Sem., 12:88–103, 2012.

[SHCvH07] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. J. Autom. Reasoning, 39(3):317–349, 2007.

[Shi94] Solomon Eyal Shimony. Finding maps for belief networks is np-hard. Artif. Intell., 68(2):399–410, 1994.

[Sin12] Parag Singla. Markov logic networks: Theory, algorithms and applications. In Proceedings of the 18th International Conference on Management of Data, COMAD 2012, 2012, Pune, India, page 15. Computer Society of India, 2012.

[SJ81] John E. Shore and Rodney W. Johnson. Properties of cross-entropy minimization. IEEE Transactions on Information Theory, 27(4):472–482, 1981.

[SK02] Steffen Schulze-Kremer. Ontologies for molecular biology and bioinformatics. In Silico Biology, 2(3):179–193, 2002.

[SKC94] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1, pages 337–343. AAAI Press / The MIT Press, 1994.

[SKH11] Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. Consequence-based reasoning beyond horn ontologies. In IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, pages 1093–1098. IJCAI/AAAI, 2011.

[SKLG08] Parag Singla, Henry Kautz, Jiebo Luo, and Andrew Gallagher. Discovery of social relationships in consumer photo collections using markov logic. In Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on, pages 1–7. IEEE, 2008.

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008, volume 432 of CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[SMSS10] Nikos Simou, Theofilos P. Mailis, Giorgos Stoilos, and Giorgos B. Stamou. Optimization techniques for fuzzy description logics. In Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Waterloo, Ontario, Canada, May 4-7, 2010, volume 573 of CEUR Workshop Proceedings, pages 244–254. CEUR-WS.org, 2010.

[Spi71] F. Spitzer. Markov random fields and gibbs ensembles. American Mathematical Monthly, pages 142–154, 1971.

[SS90] Glenn Shafer and Prakash P. Shenoy. Probability propagation. Ann. Math. Artif. Intell., 2:327–351, 1990.

[ŠSB+05] Ondrej Šváb, Vojtech Svátek, Petr Berka, Dušan Rak, and Petr Tomášek. Ontofarm: Towards an experimental collection of parallel ontologies. Poster Track of ISWC, 2005.

[SSP+07] Giorgos Stoilos, Giorgos B. Stamou, Jeff Z. Pan, Vassilis Tzouvaras, and Ian Horrocks. Reasoning with very expressive fuzzy description logics. J. Artif. Intell. Res. (JAIR), 30:273–320, 2007.

[SSS13] Nikos Simou, Giorgos Stoilos, and Giorgos B. Stamou. Storing and querying fuzzy knowledge in the semantic web using fire. In Uncertainty Reasoning for the Semantic Web II, International Workshops URSW 2008-2010 Held at ISWC and UniDL 2010 Held at FLoC, Revised Selected Papers, volume 7123 of Lecture Notes in Computer Science, pages 158–176. Springer, 2013.

[SSSK06] Giorgos Stoilos, Nikos Simou, Giorgos B. Stamou, and Stefanos D. Kollias. Uncertainty and the semantic web. IEEE Intelligent Systems, 21(5):84–87, 2006.

[Str05a] Umberto Straccia. Description logics with fuzzy concrete domains. In UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005, pages 559–567. AUAI Press, 2005.

[Str05b] Umberto Straccia. Towards a fuzzy description logic for the semantic web (preliminary report). In The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings, volume 3532 of Lecture Notes in Computer Science, pages 167–181. Springer, 2005.

[Str06] Umberto Straccia. Chapter 7 uncertainty and description logic programs over lattices. In Elie Sanchez, editor, Fuzzy Logic and the Semantic Web, volume 1 of Capturing Intelligence, pages 115 – 133. Elsevier, 2006.

[Stu08] Heiner Stuckenschmidt. Debugging owl ontologies - a reality check. In Proceedings of the 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON-SWSC-2008), Tenerife, Spain, June 1-2, 2008, volume 359 of CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[T+13] Divya Thomas et al. A survey on various ontology debugging methods. International Journal of Advanced Research in Electronics and Communication Engineering, 2(3):pp–289, 2013.

[TCKG05] Benjamin Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: a large margin approach. In Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005, volume 119 of ACM International Conference Proceeding Series, pages 896–903. ACM, 2005.

[TD08] Son Dinh Tran and Larry S. Davis. Event modeling and recognition using markov logic networks. In Computer Vision - ECCV 2008,

10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part II, volume 5303 of Lecture Notes in Computer Science, pages 610–623. Springer, 2008.

[TFL+10] Matthias Thimm, Marc Finthammer, Sebastian Loh, Gabriele Kern-Isberner, and Christoph Beierle. A system for relational probabilistic reasoning on maximum entropy. In Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference, May 19-21, 2010, Daytona Beach, Florida. AAAI Press, 2010.

[TV03] Christoph Tempich and Raphael Volz. Towards a benchmark for semantic web reasoners - an analysis of the daml ontology library. In EON2003, Evaluation of Ontology-based Tools, Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools held at the 2nd International Semantic Web Conference ISWC 2003, 20th October 2003 (Workshop day), Sundial Resort, Sanibel Island, Florida, USA, volume 87 of CEUR Workshop Proceedings. CEUR-WS.org, 2003.

[VG12] Deepak Venugopal and Vibhav Gogate. On lifting the gibbs sampling algorithm. In Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, pages 1664–1672, 2012.

[VN11] Johanna Völker and Mathias Niepert. Statistical schema induction. In The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I, volume 6643 of Lecture Notes in Computer Science, pages 124–138. Springer, 2011.

[vR79] C. J. van Rijsbergen. Information Retrieval. Butterworth, 1979.

[VVSH07] Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning disjointness. In The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings, volume 4519 of Lecture Notes in Computer Science, pages 175–189. Springer, 2007.

[WF01] Yair Weiss and William T. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. IEEE Transactions on Information Theory, 47(2):736–744, 2001.

[WG04] Wayne L Winston and Jeffrey B Goldberg. Operations research: applications and algorithms. Thomson/Brooks/Cole Belmont, 2004.

[Wil99] Hilary Paul Williams. Model building in mathematical programming. Wiley, 1999.

[WLB12] Wilson Wong, Wei Liu, and Mohammed Bennamoun. Ontology learning from text: A look back and into the future. ACM Comput. Surv., 44(4):20, 2012.

[WML+02] E. Wright, S. Mahoney, K. Laskey, M. Takikawa, and T. Levitt. Multi-entity bayesian networks for situation assessment. In Proceedings of the Fifth International Conference on Information Fusion, volume 2, pages 804–811, July 2002.

[Wol00] Laurence A Wolsey. Integer programming. IIE Transactions, 32(273-285):2–58, 2000.

[WW08] Fei Wu and Daniel S. Weld. Automatically refining the wikipedia infobox ontology. In Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008, pages 635–644. ACM, 2008.

[XW07] Neng-fu Xie and Wen-sheng Wang. Ontology and acquiring of agriculture knowledge. Agriculture Network Information, 8:13–14, 2007.

[YC05] Yi Yang and Jacques Calmet. Ontobayes: An ontology-driven uncertainty model. In 2005 International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA 2005), International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2005), 28-30 November 2005, Vienna, Austria, pages 457–463. IEEE Computer Society, 2005.

[Zho07] Lina Zhou. Ontology learning: state of the art and open issues. Information Technology and Management, 8(3):241–252, 2007.

[ZNSS11] Cäcilia Zirn, Mathias Niepert, Heiner Stuckenschmidt, and Michael Strube. Fine-grained sentiment analysis with structural features. In Fifth International Joint Conference on Natural Language Processing, IJCNLP 2011, Chiang Mai, Thailand, November 8-13, 2011, pages 336–344. The Association for Computer Linguistics, 2011.

Index