Technische Universität Hamburg-Harburg

Diplomarbeit

Analysis of the average-case behavior of an inference algorithm for probabilistic description logics

submitted by
Tobias H. Näth, ZD (B.Sc.)

Supervisors:

Prof. Dr. Ralf Möller
Prof. Dr.-Ing. Rolf-Rainer Grigat
M.Sc. Irma Sofia Espinosa Peraldi

Institute of Software, Technology & Systems (STS)
Technical University Hamburg–Harburg
Hamburg, GERMANY

February 2007

DEDICATED TO MY BROTHER

Thanks to

Professor Ralf Möller, for the challenging and interesting topic, his time and personal support.

Irma Sofia Espinosa Peraldi, for her patience, suggestions and personal support.

Morten Vierling and Christine Kaland, for a deciding tip that helped to cross the Rubicon.

Christina Wenterodt, for her outstanding illustrations.

Martin Malzahn, Mirko Heger and Volker Lübbers, for their friendship, suggestions and support.

Thomas L., Annemarie and Volker G. Näth, for their love and unconditional support

Hamburg, February 6, 2007

Hiermit erkläre ich, Tobias H. Näth (Matrikelnummer 18530), daß ich die von mir am heutigen Tage dem Prüfungsausschuß des Studienganges Informatik-Ingenieurwesen an der Technischen Universität Hamburg-Harburg eingereichte Diplomarbeit zum Thema

"Analysis of the average-case behavior of an inference algorithm for probabilistic description logics"

vollkommen selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Tobias H. Näth

Contents

Chapter 1

Introduction

How to introduce an abstract topic to a reader so that he or she is captured easily? You start with telling a story or two.

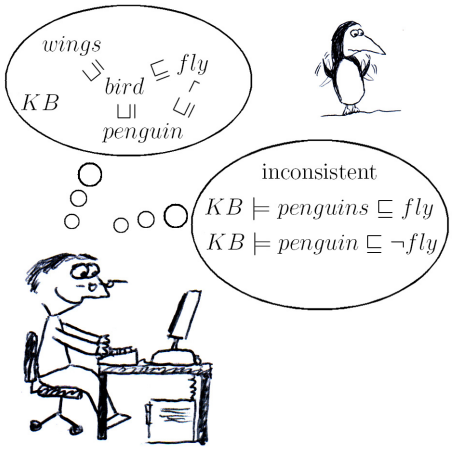
As good stories always start:

1.1 Contra Bovem Rufum

Once upon a time there was a computer scientist who wanted to capture the world of birds in a knowledge base. In a knowledge base he stores statements about the world. Afterwards he wants to apply reasoning techniques onto his knowledge base in order to access knowledge which is implied within the statements stored.

So our scientist starts out to model the world of birds. He stores the statements:

BIRDS FLY ($bird \sqsubseteq fly$) and BIRDS HAVE WINGS ($bird \sqsubseteq wings$).

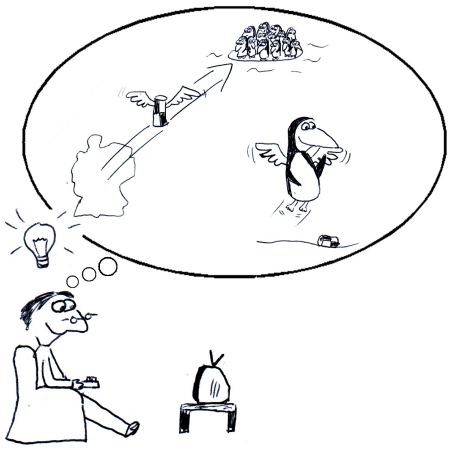


After a while he wants to model penguins in his knowledge base. So he starts by adding PENGUINS ARE BIRDS ($penguin \sqsubseteq bird$). Then he adds PENGUINS DO NOT FLY ($penguin \sqsubseteq \neg fly$).

In adding this statement our scientist realises that his knowledge base just became inconsistent. Out of an inconsistent knowledge base anything may be inferred.

His knowledge base now entails the contradiction PENGUINS FLY ($KB \models penguins \sqsubseteq fly$)

and PENGUINS DO NOT FLY ($KB \models penguin \sqsubseteq \neg fly$). So he calls it a day and goes home.



Watching television commercials at home he is inspired how to solve the problem at hand. His idea is to ship lots of energy drinks to the south pole and distribute it to the penguins. And *voila*, all penguins gain the ability to fly as suggested by a commercial.

He would be able to remove the statement, that PENGUINS DO NOT FLY and regain a consistent knowledge base.

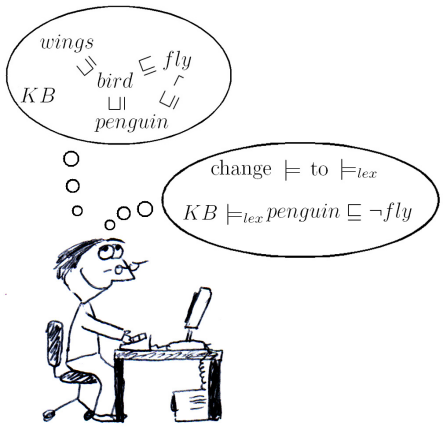
The problem solved he is now ready to go to bed. He has an untroubled sleep.

But next morning he gets his doubts. His solution has three major drawbacks:

First, it will be very expensive to purchase all these energy drinks and ship them to the south pole.

Second, penguins do not dig energy drinks.

Third, at the south pole there are no recycling centres to get rid of all the empty cans.



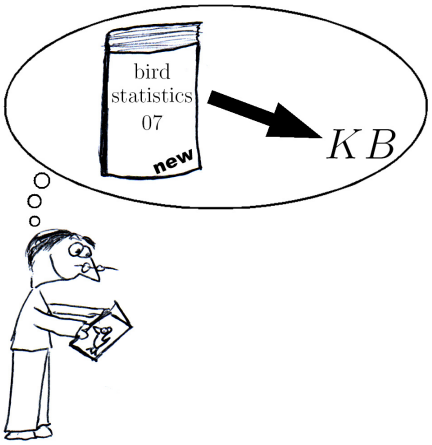
Back at work the scientist thinks about a better solution. How about changing the entailment relation in such a way, that the knowledge base only entails PENGUINS DO NOT FLY ($KB \models_{lex} penguin \sqsubseteq \neg fly$).

How to achieve such an entailment is one of the ideas presented in this thesis.

But lets continue with a second story:

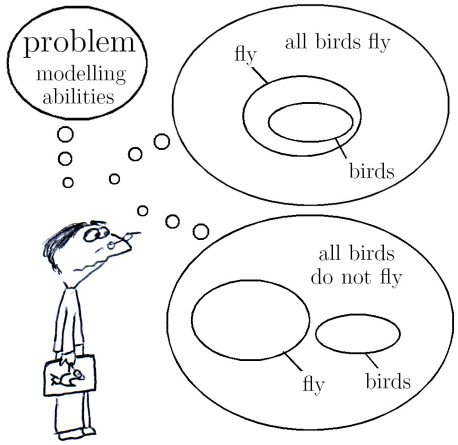
1.2 Bird Statistics 07

Once upon another time there was a computer scientist who laid his hands on the brand new book "Bird Statistics 07".



He previously worked on a knowledge base which was capturing all knowledge about birds.

So he thought there are new statements to be fed into his knowledge base.



But he realised that he got a problem.

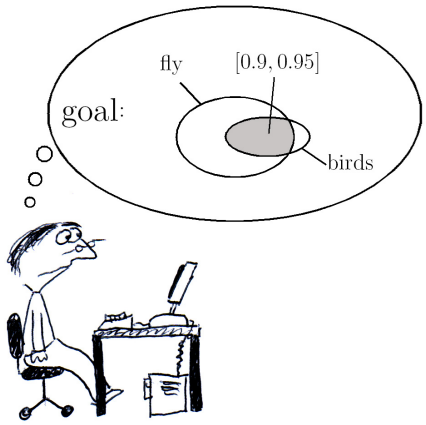
His modelling language would only allow to express statements in the following fashion:

100% OF ALL BIRDS FLY

or

0% OF ALL BIRDS FLY.

He was not able to specify some degree of overlap.



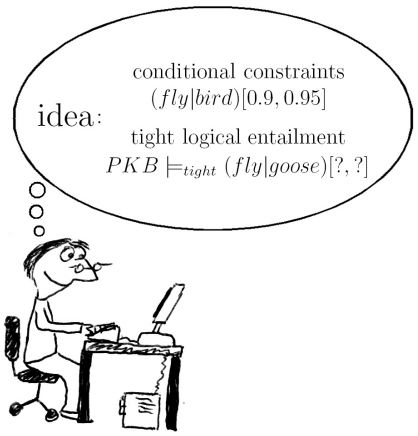
But the knowledge available in his new book was stated in this way:

90% TO 95% OF ALL BIRDS FLY.

So he was thinking about ways for extending his knowledge base in order to capture the type of statements contained in his book.

The construct he finally came up with is called *conditional constraint* and written as follows $(fly|bird)[0.9, 0.95]$.

This construct is tightly related to conditional probabilities. Therefore we call the first concept, here fly, *conclusion* and the second, here *bird*, *evidence*.



In order to draw inferences from this kind of knowledge he came up with the idea of *tight logical entailment*. This entailment computes the tightest probability bounds on given *evidence* and *conclusion*. For example he could be interested in knowing the tightest probability bounds for GEESE TO FLY ($PKB \models_{tight} (fly|goose)[?, ?]$).

This thesis deals with the combination of the views presented here and in the previous section 1.1.

Hopefully you eagerly want to read this thesis now. But you might argue: "What about the computer scientist?" I tell you: "He lived happily ever after."

1.3 Thesis Statement

The objective of this thesis is to investigate an extension of Description Logics with probabilistic lexicographic entailment proposed by Giugno and Lukasiewicz [24].

To achieve this objective, a prototypical implementation in software of the proposed algorithm for this extension will be attempted. This is done in order to show that the approach proposed can be implemented. Using the implementation of the algorithm an average case analysis is performed for proving its usability.

The implementation of the algorithm requires the integration of a reasoner for expressive description logics and of a linear program solver.

1.4 Overview

In Chapter 2 the preliminaries for this thesis are given. The reader will find short introductions to topics which build the base of the theories presented in Chapter 3. He or she will also find hints for further reading.

Chapter 3 presents an overview of related research and the theory behind probabilistic lexicographic entailment. In addition elaborate examples should guide the reader.

The analysis of the prototypical requirements, its architecture, design and implementation are laid out in Chapter 4.

Chapter 5 is dedicated to the average case analysis of the proposed inference algorithm. Results of the analysis and ideas on how to improve the algorithms performance will be shown.

In Chapter 6 the achievements of this thesis are summarised, conclusions are drawn and recommendations for future research and development are provided.

Chapter 2

Preliminaries

Here the foundations of the theory, presented in Chapter 3, are introduced. It is **not** intended to do complete and comprehensive discussions of these topics.

2.1 Introduction to Linear Programming

Linear Programming in a "mathematical nutshell" is the solution of finite systems of linear equations and inequalities in a finite number of variables [46].

A linear program is defined as follows:

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j \tag{2.1a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_j^i x_j \geq b_i \quad \text{for } i = 1, \ldots, m \tag{2.1b}$$

$$x_j \geq 0 \quad \text{for } j = 1, \ldots, m \tag{2.1c}$$

Formula 2.1a is called the *objective function* of the linear program. The goal is to compute its maximum (respectively minimum) with respect to the set of linear constraints given in 2.1b. Usually the considered variables x_j have to be non-negative (2.1c).

There are three possible solutions to a linear program:

1. the solution is unfeasible, meaning the linear program has no solution at all
2. the solution is unbounded, meaning the solution goes to infinity
3. the solution is the optimal one.

Linear programs may be applied to a wide range of problems in planning and scheduling, e.g. in business and economics.

2.1.1 Solving linear programs

In the late 1940's, George B. Dantzig [2], a mathematician at the US Air Force, invented the so called *simplex method* [16]. This invention started off research in linear programming and its applications, which ignited the field of operations research. In 1972 the worst case behaviour of the algorithm was shown by Klee and Minty to have exponential complexity. Ten years later Borgwardt was able to proof that its average case behaviour is of polynomial complexity [13]. In recent years, the simplex method performance has been still a matter of research interest [58].

A new algorithm was proposed in 1979, which owns polynomial worst case behaviour [46]. It is called the *ellipsoid algorithm*. However, this algorithm has convergence problems due to numerical computation.

The *interior point algorithm* was devised in 1984, providing both polynominal complexity and numerical stability. Nevertheless improved simplex algorithms remain used widely in practical applications.

Software packages implementing algorithms to solve linear programs are called *linear program solver* (see also subsection 4.1.3).

2.1.2 Further Reading

There are numerous textbooks available on linear programming. The author had had a look at [46, 13, 20].

2.2 Introduction to Description Logics

Description logics in a mathematical nutshell are proven computable subsets of first order logic. Previously this field of research was known under the terms *Concept Languages* and *Terminological Systems*. Today *Description Logics* is the term most commonly used.

Description Logics are a family of languages, which are used to model a certain application domain. All family members are based on the two central ideas of *atomic concepts* and *roles*. Looking from first-order logic point of view, concepts are unary and roles are binary predicates to represent relations between concepts.

For example *bird*, fly, *wings* and *penguins* are *atomic concepts* and *hasChild* is an *atomic role*.

The languages differ by their constructors, which are provided to construct complex concepts and roles out of atomic ones. This is also true for extensions, which may introduce notions of time or notions of uncertainty into the given logic.

2.2.1 Knowledge Representation

A Description Logic Language may be used for representing statements which are stored into and retrieved from a knowledge base. Generally a knowledge base is a storage for knowledge.

You have already seen one of these language constructs (see section 1.1) in the introduction, for example: $bird \sqsubseteq fly$.

It is called *inclusion axiom* and states that the concept *bird* is a subset of the concept fly.

Further it is allowed to state concept definitions in order to construct more complex concepts, for example: $hen \equiv bird \sqcap female$.

Concept definitions and inclusion axioms naturally form a hierarchy of concepts, the so called *terminology* or TBox. There exists a highest concept in any given hierarchy called *top concept* or short *top*, denoted $\top$. All other concepts are subsumed by $\top$; this means that all other concepts are subsets of $\top$.

Correspondingly there is a lowest concept in any given hierarchy called *bottom concept* or *bottom*, it is denoted by the character $\bot$. This concept is equal to the empty set, commonly denoted $\{\}$ or $\emptyset$. Concepts subsumed by bottom are equally empty.

There exists a second part of description logic knowledge base, the ABox, which stores so called *assertional knowledge*. This is knowledge about the elements contained in the concepts. The elements are also called *individuals*.

Statements stored in an ABox are called *membership assertions*. An example for such a membership assertion would be: $female \sqcap penguin(GLORIA)$.

The *unique name assumption* is commonly applied to ABoxes. With the assumption it is expressed that each individual has a unique name.

TBox and ABox constitute the knowledge base for reasoning systems based on description logics.

2.2.2 Inference

The purpose of a knowledge base is not only to retrieve the stored knowledge, but also to draw conclusions out of it. For example we would like to compute, if our knowledge base entails that:

1. PENGUINS HAVE WINGS ($KB \models penguins \sqsubseteq wings$)
2. GLORIA IS A HEN ($KB \models hen(Gloria)$)

The main reasoning services with respect to a TBox are *subsumption* and *satisfiability*. Subsumption is a test for a subset relationship between two concepts (see 1. example above). Satisfiability computes if a concept contains possible individuals and therefore is not equal to $\bot$

With respect to an ABox the main reasoning service is instance checking (see 2. example above).

Research on Description Logics has provided us with algorithms to carry out these inferences. It has been shown, that computational complexity of the inferences depend on the expressiveness of the selected Description Logic. Several algorithms have been implemented in Systems called *Description Logic Reasoner*, which are presented in subsection 4.1.2.

Usually inferences are carried out under the *open-world assumption*. The assumption says that an algorithm finds a prove for the reasoning task at hand or it concludes that the knowledge in the knowledge base is insufficient. This does not mean that the statement requested is false.

2.2.3 Further Reading

If the reader is relatively new to the topic of Description Logics, the author would like to refer him to the comprehensive Description Logic Handbook [11] and recommends to have a look at formal introduction to a description logic language (like subsection 2.2.1 in [11]) before resuming this thesis. Readers interested in $\mathcal{SHOQ}(\mathbf{D})$ should have a look at [29].

Chapter 3

Probabilistic Description Logics

This Chapter gives an overview over the research related and presents two theories. The theory of *Lexicographic entailment*, presented in section 3.2, provides the background to the "Contra Bovem Rufum" story in the introduction. Section 3.3 combines the background of both stories presented into one theory called *Probabilistic Lexicographic Entailment for Description Logics*.

3.1 Related Research

The research related to the theories under investigation may be divided into two categories: *probability logics* on one hand and *non-monotonic logics* on the other. The first category deals with the combination of probability theory and logic, the second with exceptions in the logic.

3.1.1 Logics & Probabilities

In "Boole's Logic and Probability", Halperin [28] investigates the research George Boole(1815-1864) did on the combination of probability theory and logic. He also describes, how Boole's work may be applied to the subject of linear programming. Thus we may state that combining logics and probability is rather an old field of research.

Probabilistic Reasoning

Several probabilistic reasoning methods for knowledge based systems have been discussed in books by Pearl[48], Paass [45], Bacchus [12], and Russel and Norvig[55]. In the literature

two approaches can be found in order to perform inferences with probabilities. The first approach uses Bayesian or believe networks to determine the inferred probabilities [48, 55], the second makes use of linear programming for this task [45].

Probabilistic extensions for Description Logics

Both approaches previously mentioned have been tried to be combined with description logics. The usage of Bayesian networks has been applied in [33, 17] in order to achieve the probabilistic extension. In [32] the inference could possibly be made by linear programming. The paper under investigation [24] and its follow up [39] are using linear programming for probabilistic inference.

3.1.2 Non-monotonic Logics

Both theories, which will be introduced in the next sections, may be categorised as non-monotonic logics. The difference between monotonic and non-monotonic reasoning is summarised below.

Monotonic Reasoning

Reasoning in Predicate or First Order Logic is monotone with respect to the available knowledge. More precisely adding new statements to the knowledge base "can only increase the set of entailed sentences" [55] or in other words "if something is known to be *true* (*false*), it will never become *false* (*true*)" [27]. In [55]]this is formally stated as:

$$\text{If} \qquad KB \models \phi \qquad \text{then} \qquad KB \wedge \psi \models \phi \tag{3.1}$$

Non-monotonic Reasoning

By the observation of human reasoning one recognises that it is not monotone at all. This can be informally described as "changing one's mind". Humans allow exceptions in their knowledge which can not be resembled neither in Predicate nor in First Order Logic. To address these shortcomings several non-monotonic reasoning methods [52, 42, 43] have been developed.

Default Logic is a non-monotonic reasoning method developed by Reiter [52]. It is also discussed in [22, 48, 55]. In this approach one differentiates between hard facts about a world and default rules. "Since the set of all hard facts cannot completely specify the

world, we are left with gaps in our knowledge; the purpose of the default rules is to help fill in those gaps with plausible conclusions" [48].

Further research in default logic developed several different kinds of entailments, e.g. 0-entailment, 1-entailment, z-entailment, lexicographic entailment and me-entailment to name a few (see [15] for an overview).

Lexicographic entailment[35] and its probabilistic successors[40, 37, 25, 39] will be the object of theory investigation in this thesis.

3.2 Lexicographic Entailment

The notion of *lexicographic entailment* was first proposed by Lehmann [35] for sets of defaults. As an extension of Pearls System Z [49] it allows for inheritance to exceptional "subclasses".

To get a better understanding of the discussed topic let us reintroduce the example of story 1.1, which is called "Winged Penguins" in literature [35, 15]. It is also used to illustrate the terms needed. The syntax is the one used by Lehmann.

Consider the following set of defaults D:

$$\mathrm{D} = \{(b : f), (b : w), (p : b), (p : \neg f)\} \tag{3.2}$$

This set of defaults D may be interpreted as follows:

BIRDS(b)FLY(f), BIRDS HAVE WINGS(w), PENGUINS(p) ARE BIRDS BUT PENGUINS DO NOT FLY.

Defaults represent knowledge about how things generally are, i.e *birds fly*. Lehmann chooses the presumptive reading of a default to give its meaning: "Birds are presumed to fly unless there is evidence to the contrary." He denotes defaults as $(\phi : \psi)$ where ϕ and ψ are formulae in propositional logic. The meaning of a default $(\phi : \psi)$ is closely related to the implication $\phi \Rightarrow \psi$, also called the defaults material counterpart. The interpretation is given as statements of high conditional probability $P(\psi|\phi) \geq 1 - \epsilon$ [49].

If we would have used a propositional knowledge base in our example instead of defaults it would be inconsistent and therefore entail anything, e.g. penguins fly and penguins don't fly. The goal of the definition of the lexicographic entailment is to regain a consistent knowledge base, a knowledge base where we only are able to conclude that penguins don't fly.

We achieve this goal by applying a lexicographic ordering onto our models in order to

m	b	f	p	w	$b \Rightarrow f$	$b \Rightarrow w$	$p \Rightarrow b$	$p \Rightarrow \neg f$	D_0	D_1	$p \wedge w$	$p \wedge \neg w$
m_1	0	0	0	0	1	1	1	1	2	2	0	0
m_2	0	0	0	1	1	1	1	1	2	2	0	0
m_3	0	0	1	0	1	1	0	1	2	1	0	1
m_4	0	0	1	1	1	1	0	1	2	1	1	0
m_5	0	1	0	0	1	1	1	1	2	2	0	0
m_6	0	1	0	1	1	1	1	1	2	2	0	0
m_7	0	1	1	0	1	1	0	0	2	0	0	1
m_8	0	1	1	1	1	1	0	0	2	0	1	0
m_9	1	0	0	0	0	0	1	1	0	2	0	0
m_{10}	1	0	0	1	0	1	1	1	1	2	0	0
m_{11}	1	0	1	0	0	0	1	1	0	2	0	1
m_{12}	1	0	1	1	0	1	1	1	1	2	1	0
m_{13}	1	1	0	0	1	0	1	1	1	2	0	0
m_{14}	1	1	0	1	1	1	1	1	2	2	0	0
m_{15}	1	1	1	0	1	0	1	0	1	1	0	1
m_{16}	1	1	1	1	1	1	1	0	2	1	1	0

Table 3.1: Winged Penguins

consider only preferred or more natural models. We give reasons for doing so shortly, but for now let's return to the "Winged Penguins" example.

In propositional logic a truth assignment to a primitive proposition is called a *model* (or *world*). As our example shows we have four primitive propositions b, f, p and w. Because of that there are 2^4 possible models denoted m_i with $i \in \{1, \ldots, 16\}$. See table 3.1[1] for further insight.

A model m *satisfies* a propositional formula ϕ if and only if $m(\phi) = true$. This is represented by $m \models \phi$. A default $(\phi : \psi)$ is satisfied by a model, denoted $m \models (\phi : \psi)$, if and only if $m \models \phi \Rightarrow \psi$.

Further a model is said to *verify* (or as the opposite *falsify*) a default $(\phi : \psi)$ if and only if $m \models \phi \wedge \psi$ (or $m \models \phi \wedge \neg\psi$). One default is *tolerated* by a set of defaults D iff there exists a model which verifies it and simultaneously satisfies the others. If no such model exists, the default is said to be *in conflict* with the other defaults.

In the example one can see that the default $(b : f)$ is tolerated by D. For that purpose we first have a look at the models $m_{13} - m_{16}$ which verify the default. Afterwards m_{14} can

[1] In this table 0 represents $false$ and 1 represents $true$; except for the columns D_0 and D_1, they represent the count of satisfied defaults in the subset D_i of the z-partition.

be identified to satisfy all remaining defaults. m_{14} also shows that the default $(b : w)$ is tolerated by D. If we inspect the models $m_{11}, m_{12}, m_{15}, m_{16}$, which are the ones verifying the default $(p : b)$, then none of them satisfies all other three defaults at the same time. Therefore $(p : b)$ is in conflict with D. Similarly the reader may show the conflict of $(p : \neg f)$.

The *z-partition* of the set of defaults D is defined as the unique ordered partition $(\mathrm{D}_0, \ldots, \mathrm{D}_k)$ such that each D_i with $i \in \{0, \ldots, k\}$ contains the set of all defaults that are tolerated under $\mathrm{D} \setminus (\mathrm{D}_0 \cup \cdots \cup \mathrm{D}_{i-1})$.

Above we did the first step for computing the z-partition of our example. Only the defaults $(b : f)$ and $(b : w)$ were tolerated by all other defaults. Thus they build D_0. D_0 is then removed from D. The remaining two defaults tolerate each other. Hence they are contained in D_1. We now have the following z-partition $(\mathrm{D}_0, \mathrm{D}_1)$ where:

$$\mathrm{D}_0 = \{(b : f), (b : w)\} \qquad \text{and} \qquad \mathrm{D}_1 = \{(p : b), (p : \neg f)\} \tag{3.3}$$

If the unique z-partition can be build the set of defaults D is said to be *consistent*.

The z-partition groups the defaults by specificity. A part of the z-partition is more specific than another if its index is higher. For example the defaults $(p : b)$, $(p : \neg f)$ in part D_1 are the most specific ones.

With possible conflicting defaults and the inability to find a model that tolerates all of them, the goal is to select a preferred model to draw the conclusion from. Because of this two criteria have to be taken into account:

1. A model is preferred to another model if it satisfies more defaults.
2. Satisfying a more specific default is preferred to a less specific one.

Both criteria allow us to order the models according to them. Both orderings should be combined into one, in order to determine the preferred model. A lexicographic ordering provides such an integration of orderings. The second criterion is chosen as the major criterion, since it is preferred to satisfy a more specific default than to satisfy less specific ones.

Using the z-partition as a degree of specificity the lexicographic order for the models is defined as follows:

A model m is called *lexicographical preferred* to a model m' if and only if some $i \in \{0, \ldots, k\}$ can be found that $|\{d \in \mathrm{D}_i \mid m \models d\}| > |\{d \in \mathrm{D}_i \mid m' \models d\}|$ and $|\{d \in \mathrm{D}_j \mid m \models d\}| = |\{d \in \mathrm{D}_j \mid m' \models d\}|$ for all $i < j \leq k$.

Or in other words we are counting the defaults satisfied in each part of the z-partition for every model. Then starting with the k th subset of the z-partition and comparing which model satisfies more defaults d in that subset. If there can not be made a decision we have to look at the $k-1$ subset and make the comparison and so on.

In the example we have the following lexicographic order on the models: $m_1, m_2, m_5, m_6, m_{14} > m_{10}, m_{12}, m_{13} > m_9, m_{11} > m_3, m_4, m_{16} > m_{15} > m_7, m_8$.

m is said to be the *lexicographically minimal* model if and only if no other models are lexicographically preferable to m. Lexicographic entailment is defined by comparing the minimal verifying and falsifying models of a default $(\phi : \psi)$.

In order to determine if the default $(p : w)$, which can be interpreted as PENGUINS HAVE WINGS, is lexicographically entailed in D, one has to look at the models that verify or falsify the default $(p : w)$ and therefore have to satisfy $p \wedge w$ and do not satisfy $p \wedge \neg w$ or vice versa, here the models $m_3, m_4, m_7, m_8, m_{11}, m_{12}, m_{15}, m_{16}$. The lexicographically minimal model is m_{12}. This model verifies the default $(p : w)$ and therefore PENGUINS HAVE WINGS is lexicographically entailed in D.

3.3 Probabilistic Lexicographic Entailment for Description Logics

In order to extend a Description Logic for dealing with probabilistic knowledge an additional syntactical and semantical construct is needed. This additional construct is called a *conditional constraint*. It is an extension of defaults, introduced in the previous section, which represent knowledge that is generally true.

This extension of description logics has been first formalized by Giugno and Lukasiewicz in 2002 [24, 25]. All following definitions are given in accordance to their paper.

A conditional constraint consists of a statement of conditional probability for two concepts C, D as well as a lower bound l and an upper bound u constraint on that probability. It is written as follows:

$$(D|C)[l, u] \tag{3.4}$$

Where C can be called the *evidence* and D the *hypothesis*

To gain the ability to store such statements in a knowledge base it has to be extended to a probabilistic knowledge base PKB here named probabilistic terminology $\mathcal{P}$. Additionally to the TBox T here classical terminology $\mathcal{T}_g$ of a description logic knowledge base we introduce the PTBox PT here generic probabilistic terminology $\mathcal{P}_g$, which consists of

$\mathcal{T}_g$ and a set of conditional constraints $\mathcal{D}_g$, and a PABox P_o or assertional probabilistic terminology $\mathcal{P}_o$ holding conditional constraints for every probabilistic individual o. In [25] there is no ABox declared, knowledge about so called classical individuals is also stored inside the TBox abusing nominals.

$\mathcal{D}_g$ therefore represents statistical knowledge about concepts and P_o represents degrees of belief about the individuals o.

To be able to define the semantics for a Description Logic with probabilistic extension the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot)$ has to be extended by a probability function μ on the domain of interpretation $\Delta^{\mathcal{I}}$. The extended interpretation is called the *probabilistic interpretation* $\mathcal{P}r = (\mathcal{I}, \mu)$. Each individual o in the domain $\Delta^{\mathcal{I}}$ is mapped by the probability function μ to a value in the interval [0,1] and the values of all $\mu(o)$ have to sum up to 1 for any probabilistic interpretation $\mathcal{P}r$.

With the probabilistic interpretation $\mathcal{P}r$ at hand the probability of a concept C, represented by $\mathcal{P}r(C)$, is defined as sum of all $\mu(o)$ where $o \in C^{\mathcal{I}}$.

The probabilistic interpretation of a conditional probability $\mathcal{P}r(D|C)$ is given as

$$\frac{\mathcal{P}r(C \sqcap D)}{\mathcal{P}r(C)} \tag{3.5}$$

where $\mathcal{P}r(C) > 0$.

In order to achieve a probabilistic lexicographic entailment all the terms known from the previous section 3.2 have to be redefined for the context of Description Logics.

A conditional constraint $(D|C)[l, u]$ is *satisfied* by $\mathcal{P}r$ or $\mathcal{P}r$ *models* $(D|C)[l, u]$ if and only if $\mathcal{P}r(D|C) \in [l, u]$. We will write this as $\mathcal{P}r \models (D|C)[l, u]$. A probabilistic interpretation $\mathcal{P}r$ is said to *satisfy* or *model* a terminology axiom T, written $\mathcal{P}r \models T$, if and only if $\mathcal{I} \models T$. A set $\mathcal{F}$ consisting of terminological axioms and conditional constraints, where F denotes the elements of $\mathcal{F}$, is satisfied or modeled by $\mathcal{P}r$ if and only if $\mathcal{P}r \models F$ for all $F \in \mathcal{F}$.

The *verification* of a conditional constraint $(D|C)[l, u]$ is defined as $\mathcal{P}r(C) = 1$ and $\mathcal{P}r$ has to be a model of $(D|C)[l, u]$. We also may say $\mathcal{P}r$ *verifies* the conditional constraint $(D|C)[l, u]$. On the contrary the *falsification* of a conditional constraint $(D|C)[l, u]$ is given if and only if $\mathcal{P}r(C) = 1$ and $\mathcal{P}r$ does **not** satisfy $(D|C)[l, u]$. It is also said that $\mathcal{P}r$ *falsifies* $(D|C)[l, u]$.

Further a conditional constraint F is said to be *tolerated* under a Terminology $\mathcal{T}$ and a set of conditional constraints $\mathcal{D}$ if and only if a probabilistic interpretation $\mathcal{P}r$ can be found that verifies F and $\mathcal{P}r \models \mathcal{T} \cup \mathcal{D}$.

With all these definitions at hand we are now prepared to define the *z-partition* of a

set of generic conditional constraints $\mathcal{D}_g$. The z-partition is build as ordered partition $(\mathcal{D}_0, \ldots, \mathcal{D}_k)$ of $\mathcal{D}_g$, where each part $\mathcal{D}_i$ with $i \in \{0, \ldots, k\}$ is the set of all conditional constraints $F \in \mathcal{D}_g \setminus (\mathcal{D}_0 \cup \cdots \cup \mathcal{D}_{i-1})$, that are tolerated under the generic terminology $\mathcal{T}_g$ and $\mathcal{D}_g \setminus (\mathcal{D}_0 \cup \cdots \cup \mathcal{D}_{i-1})$.

If the z-partition can be build from a generic probabilistic terminology $\mathcal{P}_g = (\mathcal{T}_g, \mathcal{D}_g)$, it is said to be *generically consistent* or *g-consistent*. A probabilistic terminology $\mathcal{P} = (\mathcal{P}_g, (\mathcal{P}_o)_{o \in I_p})$ is *consistent* if and only if $\mathcal{P}_g$ is g-consistent and $\mathcal{P}r \models \mathcal{T}_g \cup \mathcal{T}_o \cup \mathcal{D}_o \cup \{(\{o\}|\top)[1,1]\}$ for all $o \in I_p$.

Once more we use the z-partition for the definition of the lexicographic order on the probabilistic interpretations $\mathcal{P}r$ as follows:

A probabilistic interpretation $\mathcal{P}r$ is called *lexicographical preferred* to a probabilistic interpretation $\mathcal{P}r'$ if and only if some $i \in \{0, \ldots, k\}$ can be found, that $|\{F \in \mathrm{D}_i \mid \mathcal{P}r \models F\}| > |\{F \in \mathrm{D}_i \mid \mathcal{P}r' \models F\}|$ and $|\{F \in \mathrm{D}_j \mid \mathcal{P}r \models F\}| = |\{F \in \mathrm{D}_j \mid \mathcal{P}r' \models F\}|$ for all $i < j \leq k$.

We say a probabilistic interpretation $\mathcal{P}r$ of a set $\mathcal{F}$ of terminological axioms and conditional constraints is a *lexicographically minimal model* of $\mathcal{F}$ if and only if no probabilistic interpretation $\mathcal{P}r'$ is lexicographical preferred to $\mathcal{P}r$.

By now the meaning of *lexicographic entailment* for conditional constraints from a set $\mathcal{F}$ of terminological axioms and conditional constraints under a generic probabilistic terminology $\mathcal{P}_g$ is given as:

A conditional constraint $(D|C)[l,u]$ is a *lexicographic consequence* of a set $\mathcal{F}$ of terminological axioms and conditional constraints under a generic probabilistic terminology $\mathcal{P}_g$, written as $\mathcal{F} \mid\!\sim (D|C)[l,u]$ under $\mathcal{P}_g$, if and only if $\mathcal{P}r(D) \in [l,u]$ for every lexicographically minimal model $\mathcal{P}r$ of $\mathcal{F} \cup \{(C|\top)[1,1]\}$. *Tight lexicographic consequence* of $\mathcal{F}$ under $\mathcal{P}_g$ is defined as $\mathcal{F} \mid\!\sim_{tight} (D|C)[l,u]$ if and only if l is the infimum and u is the supremum of $\mathcal{P}r(D)$. We define $l = 1$ and $u = 0$ if **no** such probabilistic interpretation $\mathcal{P}r$ exists.

Finally we define lexicographic entailment using a probabilistic terminology $\mathcal{P}$ for generic and assertional conditional constraints F.

If F is a generic conditional constraint, then it is said to be a lexicographic consequence of $\mathcal{P}$, written $\mathcal{P} \mid\!\sim F$ if and only if $\emptyset \mid\!\sim F$ under $\mathcal{P}_g$ and a tight lexicographic consequence of $\mathcal{P}$, written $\mathcal{P} \mid\!\sim_{tight} F$ if and only if $\emptyset \mid\!\sim_{tight} F$ under $\mathcal{P}_g$.

If F is an assertional conditional constraint for $o \in I_P$, then it is said to be a lexicographic consequence of $\mathcal{P}$, written $\mathcal{P} \mid\!\sim F$, if and only if $\mathcal{T}_o \cup \mathcal{D}_o \mid\!\sim F$ under $\mathcal{P}_g$ and a tight lexicographic consequence of $\mathcal{P}$, written $\mathcal{P} \mid\!\sim_{tight} F$ if and only if $\mathcal{T}_o \cup \mathcal{D}_o \mid\!\sim_{tight} F$ under $\mathcal{P}_g$.

3.3.1 Computing tight lexicographic consequence

Now we will introduce the techniques required to compute g-consistency and tight lexicographic consequence by means of an example. The general approach in solving these two problems is to use standard reasoning provided by a description logic reasoner in order to build linear programs which may be handed over to a solver in order to decide the solvability of the programs. Let us consider the following Terminology $\mathcal{T}_g$ and set of conditional constraints $\mathcal{D}_g$:

$$\mathcal{T}_g = \{P \sqsubseteq B\} \qquad \mathcal{D}_g = \{(W|B)[.95, 1]; (F|B)[.9, .95]; (F|P)[0, .05]\} \tag{3.6}$$

Again we may interpret the concepts as follows: P stands for penguins, B for birds, F for flying objects and W for winged objects.

Deciding satisfiability

The first objective is to build a set $\mathcal{R}_{\mathcal{T}}(\mathcal{F})$. It contains the mappings r, which map conditional constraints $F_i = (D_i|C_i)[l_i, u_i]$ elements of a set of conditional constraints $\mathcal{F}$ onto one of the following terms $D_i \sqcap C_i$, $\neg D_i \sqcap C_i$ or $\neg C_i$ under the condition that the intersection of our mappings r is not equal to the bottom concept given the terminology $\mathcal{T}$, written $\mathcal{T} \not\models r(F_1) \sqcap \cdots \sqcap r(F_n) \sqsubseteq \bot$.

All possible mappings of our example are displayed in Table 3.2, columns 2 to 4. In the first column the intersection of the mappings r is build. As an abbreviation we will write $\sqcap r$ instead of $r(F_1) \sqcap \cdots \sqcap r(F_n)$. The rows of the first column contain a bottom concept $\bot$ incase the intersections of r are empty. Also the concepts causing the bottom concept are given. In the fifth column the $\sqcap r$ are tested against the terminology. If the intersections of r are **not** the same as the bottom concept $\bot$, this is denoted by $\models$, else it is denoted by $\bot$.

Our set $\mathcal{R}_{\mathcal{T}}(\mathcal{F})$ contains the following nine mappings r,

$\mathcal{R}_{\mathcal{T}}(\mathcal{F}) = \{$
$\{W \sqcap B, F \sqcap B, F \sqcap P\}, \{\neg W \sqcap B, F \sqcap B, F \sqcap P\},$
$\{W \sqcap B, \neg F \sqcap B, \neg F \sqcap P\}, \{\neg W \sqcap B, \neg F \sqcap B, \neg F \sqcap P\},$
$\{W \sqcap B, F \sqcap B, \neg P\}, \{\neg W \sqcap B, F \sqcap B, \neg P\},$
$\{W \sqcap B, \neg F \sqcap B, \neg P\}, \{\neg W \sqcap B, \neg F \sqcap B, \neg P\},$
$\{\neg B, \neg B, \neg P\}$
$\}$

With the set $\mathcal{R}_{\mathcal{T}}(\mathcal{F})$ at hand we are able to set up linear programs to decide the satisfiability of the terminology $\mathcal{T}$ and a finite set of conditional constraints $\mathcal{F}$. We say that

$\mathcal{T} \cup \mathcal{F}$ is satisfiable if and only if the linear program 3.7 is solvable for variables y_r, where $r \in \mathcal{R}_{\mathcal{T}}(\mathcal{F})$. This means, that in the objective function all coefficients preceding the variables y_r are set to 1.

$$\sum_{r \in R, r \models \neg D \sqcap C} -l y_r + \sum_{r \in R, r \models D \sqcap C} (1-l) y_r \geq 0 \quad \text{(for all } (D|C)[l,u] \in \mathcal{F}) \tag{3.7a}$$

$$\sum_{r \in R, r \models \neg D \sqcap C} u y_r + \sum_{r \in R, r \models D \sqcap C} (u-1) y_r \geq 0 \quad \text{(for all } (D|C)[l,u] \in \mathcal{F}) \tag{3.7b}$$

$$\sum_{r \in R} y_r = 1 \tag{3.7c}$$

$$y_r \geq 0 \quad \text{(for all } r \in R) \tag{3.7d}$$

To be able to set up linear programs we further need to introduce the meaning of $r \models C$ which is used as index of the summation in 3.7a and 3.7b. It is an abbreviation for $\emptyset \models \sqcap r \sqsubseteq C$. So the coefficient preceding the variables y_r is set in linear constraints 3.7a and 3.7b if either $r \models \neg D \sqcap C$ or $r \models D \sqcap C$ may be proven.

computing g-conscistency

With the tool at hand to decide satisfiability, we may also decide, if a conditional constraint may be tolerated by a set of conditional constraints $\mathcal{F}$. To verify a constraint we add a conditional constraint $(C|\top)[1,1]$. With the extended set the linear program is generated and solved. If an unfeasible solution is computed the conditional constraint is conflicting. If an optimal solution is found, the conditional constraint is tolerated.

Now the z-partition of a set of conditional constraints is computable. For the example we would have to set up four linear programs to compute part D_0 and two for D_1. See appendix D for the linear programs of part D_0.

computing tight logical entailment

How to compute tightest probability bounds for given evidence C and conclusion D in respect to a set of conditional constraints $\mathcal{F}$ under a terminology $\mathcal{T}$ is explained in the following text. The task is named *tight logical entailment* and denoted $\mathcal{T} \cup \mathcal{F} \models_{tight} (D|C)[l,u]$.

Given that $\mathcal{T} \cup \mathcal{F}$ is satisfiable, a linear program is set up for $\mathcal{F} \cup (C|\top)[1,1] \cup (D|\top)[0,1]$. The objective function is set to $\sum_{r \in R, r \models D} y_r$. So the coefficient in front of the variables y_r are set 1 if $r \models D$.

The tight logical entailed lower bound l is computed by minimising respectively the upper bound u by maximising the linear program.

computing probabilistic lexicographic entailment

In order to compute tight probabilistic lexicographic entailment for given evidence C and conclusion D under a generic probabilistic terminology $\mathcal{P}_\}$ and a set $\mathcal{F}$ of terminological axioms and conditional constraints the following steps have to be taken:

1. Compute the z-partition of $\mathcal{D}_g$ in order to be able to generate a lexicographic ordering
2. Compute lexicographic minimal sets $\mathcal{D}'$ of conditional constraints of $\mathcal{D}_g$ as elements of $\overline{\mathcal{D}}$.
3. Compute the tight logical entailment $\mathcal{T} \cup \mathcal{F} \cup \mathcal{D}' \models_{tight} (D|C)[l, u]$ for all $\mathcal{D}' \in \overline{\mathcal{D}}$.
4. Select the minimum of all computed lower bounds and the maximum of all upper bounds.

The 2. step needs some explanation since a new task "compute *lexicographic minimal sets*" is introduced. In order to define a lexicographic minimal set $\mathcal{D}'$, a preparatory definition is required.

A set $\mathcal{D}' \subset \mathcal{D}_g$ *lexicographic preferable* to $\mathcal{D}'' \subset \mathcal{D}_g$ if and only if some $i \in \{0, \ldots, k\}$ exists such that $|\mathcal{D}' \cap \mathcal{D}_i| > |\mathcal{D}'' \cap \mathcal{D}_i|$ and $|\mathcal{D}' \cap \mathcal{D}_i| > |\mathcal{D}'' \cap \mathcal{D}_i|$ for all $i < j \leq k$.

With the lexicographic order introduced onto the sets $\mathcal{D}'$ the definition of lexicographic minimal is given as:

$\mathcal{D}'$ is lexicographic minimal in $\mathcal{S} \subseteq \{S|S \subseteq \mathcal{D}_g\}$ if and only if $\mathcal{D}' \in \mathcal{S}$ and no $\mathcal{D}'' \in \mathcal{S}$ is lexicographic preferable to $\mathcal{D}'$.

Note that equivalence of using lexicographic minimal sets instead of lexicographically minimal models is not proven in [24]. A prove is given in the follow up [39].

3.3.2 Using probabilistic lexicographic entailment

The probabilistic lexicographic entailment may be used to decide the following two interesting probabilistic inference problems:

probabilistic subsumption

To compute a probabilistic subsumption for the concepts D, C the tight probabilistic lexicographic entailment is determined with respect to $\mathcal{P}_g$ and $\mathcal{F} = \emptyset$.

probabilistic instance checking

Probabilistic instance checking for an individual o and a concept E is done by computing the tight probabilistic lexicographic entailment with E set as conclusion and $\top$ as evidence and with respect to $\mathcal{P}_g$ and $\mathcal{F} = \mathcal{P}_o$.

$\sqcap r$	$r((W\|B)[.95,1])$	$r((F\|B)[.9,.95])$	$r((F\|P)[0,.05])$	$\mathcal{T} \not\models \sqcap r \sqsubseteq \bot$
$B \sqcap P \sqcap W \sqcap F$	$W \sqcap B$	$F \sqcap B$	$F \sqcap P$	$\models$
$B \sqcap P \sqcap \neg W \sqcap F$	$\neg W \sqcap B$	$F \sqcap B$	$F \sqcap P$	$\models$
$\bot, \neg B \sqcap B$	$\neg B$	$F \sqcap B$	$F \sqcap P$	
$\bot, \neg F \sqcap F$	$W \sqcap B$	$\neg F \sqcap B$	$F \sqcap P$	
$\bot, \neg F \sqcap F$	$\neg W \sqcap B$	$\neg F \sqcap B$	$F \sqcap P$	
$\bot, \neg B \sqcap B$	$\neg B$	$\neg F \sqcap B$	$F \sqcap P$	
$\bot, \neg B \sqcap B$	$W \sqcap B$	$\neg B$	$F \sqcap P$	
$\bot, \neg B \sqcap B$	$\neg W \sqcap B$	$\neg B$	$F \sqcap P$	
$\neg B \sqcap P \sqcap F$	$\neg B$	$\neg B$	$F \sqcap P$	$\bot$
$\bot, \neg F \sqcap F$	$W \sqcap B$	$F \sqcap B$	$\neg F \sqcap P$	
$\bot, \neg F \sqcap F$	$\neg W \sqcap B$	$F \sqcap B$	$\neg F \sqcap P$	
$\bot, \neg B \sqcap B$	$\neg B$	$F \sqcap B$	$\neg F \sqcap P$	
$B \sqcap P \sqcap W \sqcap \neg F$	$W \sqcap B$	$\neg F \sqcap B$	$\neg F \sqcap P$	$\models$
$B \sqcap P \sqcap \neg W \sqcap \neg F$	$\neg W \sqcap B$	$\neg F \sqcap B$	$\neg F \sqcap P$	$\models$
$\bot, \neg B \sqcap B$	$\neg B$	$\neg F \sqcap B$	$\neg F \sqcap P$	
$\bot, \neg B \sqcap B$	$W \sqcap B$	$\neg B$	$\neg F \sqcap P$	
$\bot, \neg B \sqcap B$	$\neg W \sqcap B$	$\neg B$	$\neg F \sqcap P$	
$\neg B \sqcap P \sqcap \neg F$	$\neg B$	$\neg B$	$\neg F \sqcap P$	
$B \sqcap \neg P \sqcap W \sqcap F$	$W \sqcap B$	$F \sqcap B$	$\neg P$	$\models$
$B \sqcap \neg P \sqcap \neg W \sqcap F$	$\neg W \sqcap B$	$F \sqcap B$	$\neg P$	$\models$
$\bot, \neg B \sqcap B$	$\neg B$	$F \sqcap B$	$\neg P$	
$B \sqcap \neg P \sqcap W \sqcap \neg F$	$W \sqcap B$	$\neg F \sqcap B$	$\neg P$	$\models$
$B \sqcap \neg P \sqcap \neg W \sqcap \neg F$	$\neg W \sqcap B$	$\neg F \sqcap B$	$\neg P$	$\models$
$\bot, \neg B \sqcap B$	$\neg B$	$\neg F \sqcap B$	$\neg P$	
$\bot, \neg B \sqcap B$	$W \sqcap B$	$\neg B$	$\neg P$	
$\bot, \neg B \sqcap B$	$\neg W \sqcap B$	$\neg B$	$\neg P$	
$\neg B \sqcap \neg P$	$\neg B$	$\neg B$	$\neg P$	$\models$

Table 3.2: $\mathcal{DL}$-Winged Penguins

Chapter 4

The *ContraBovemRufum* Prototype

Based on the defined and discussed theories, this chapter deals with the requirements analysis, architecture, design and implementation of the prototype. The resulting software implementation is attached on CD in appendix E.

4.1 Analysis

This section presents the gathered functional and non-functional requirements, short analyses of the required technologies description logic reasoner and linear program solver and a short presentation of the related nmproblog system.

4.1.1 Prototype Requirements

Functional Requirements

In order to capture the functional requirements of the intended system, the technique of Use Cases [54, 34, 21] will be applied. As the result of the analysis four Use Cases *Manage Probabilistic Knowledge Base*, *Compute Tight Lexicographic Entailment*, *Compute Probabilistic Knowledge Base Consistency* and *Compute Probabilistic Terminology Consistency* (see figure 4.1) are imposed on the prototype. They are presented in detail within tables 4.1, 4.2, 4.3 and 4.4.

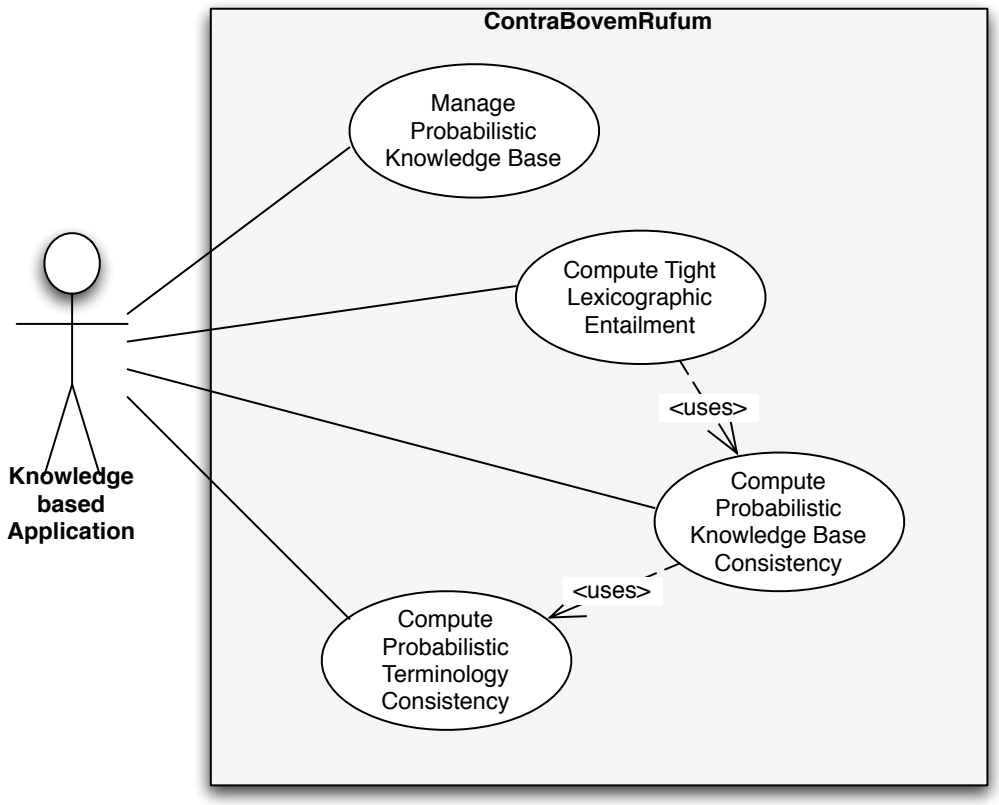


Figure 4.1: Use Case Diagram: ContraBovemRufum

Use Case:	Manage Probabilistic Knowledge Base
Summary:	A knowledge based application needs management functionality on a probabilistic knowledge base which includes creation, loading, storing and manipulation.
Actors:	Knowledge based Application
Preconditions:	The system is connected to a description logic reasoner.
Description:	1. A knowledge based application requests the creation of new probabilistic knowledge base. The system asks for a name and creates an empty probabilistic knowledge base with this name. Afterwards the knowledge base is ready to use. 2. The system is told to open a probabilistic knowledge base with a given name. If a knowledge base with the name is found, it will be loaded from persistent storage into the system and is then ready for use. [Exception: Unable to open/find probabilistic knowledge base.] 3. A request is made to save a probabilistic knowledge base with a given name. The system writes it to persistent storage. [Exception: Overwrite existing probabilistic knowledge base.] 4. A knowledge based application wants to manipulate (set, reset) the TBox or ABox of a probabilistic terminology. The system sets the desired changes. [Exception: Reasoner is not able/reachable to perform the change.] 5. The system is told to manipulate the PTBox and PABox. This means to add or remove probabilistic inclusion axioms and concept assertions.[Exception: Change could not be performed.]
Exceptions:	*Unable to open/find probabilistic knowledge base*: The application is informed that the requested probabilistic knowledge base can not be opened. *Overwrite existing probabilistic knowledge base*: The application is asked if it wishes to overwrite the probabilistic knowledge base. *Reasoner is not able/reachable to perform the change*: The application is informed why the requested action can not be performed.

	Change could not be performed: The application gets a reason for the failure
Postconditions:	none

Table 4.1: Manage Probabilistic Knowledge Base

Use Case:	Compute Tight Lexicographic Entailment
Summary:	A knowledge based Application requests the tightest lexicographic bounds on a probabilistic inclusion axiom or concept assertion with respect to a probabilistic knowledge base.
Actors:	Knowledge based Application
Preconditions:	The application provides valid evidence and conclusion concepts or as the case may be a valid concept assertion. The system is connected to a linear program solver and a description logic reasoner.
Description:	1. If the application requests to compute the tightest bounds on a probabilistic inclusion axiom, the system computes them by using evidence, conclusion, Tbox and the PTBox. [Exception: The tight lexicographic entailment fails to compute.] 2. If the tightest bounds on a probabilistic concept assertion is requested, the system computes the bounds using asserted concept, individual, TBox, ABox, PTBox and PABox. [Exception: The tight lexicographic entailment fails to compute.]
Exceptions:	*The tight lexicographic entailment fails to compute*: This may be for one of the following reasons - the request is not entailed, the system is unable to reach the solver, the system is unable to reach the reasoner. The reason is told to the application.
Postconditions:	none

Table 4.2: Compute Tight Lexicographic Entailment

Use Case:	Compute Probabilistic Knowledge Base Consistency
Summary:	On request the system checks if the probabilistic knowledge base is consistent
Actors:	Knowledge based Application
Preconditions:	The system is connected to a linear program solver and a description logic reasoner.
Description:	If the system is asked for the consistency of the probabilistic knowledge base, it first executes the Use Case Compute Probabilistic Terminology Consistency. By success of the previous task the systems checks ABox and PABox for consistency. The result is then returned to the knowledge based application.[Exception: Computation of the consistency fails.]
Exceptions:	*Computation of the consistency fails*: The application is told which part of the consistency check failed or if it failed for connectivity reasons with solver or reasoner.
Postconditions:	none

Table 4.3: Compute Probabilistic Knowledge Base Consistency

Use Case:	Compute Probabilistic Terminology Consistency
Summary:	A knowledge based application demands the PTBox consistency to be checked
Actors:	Knowledge based Application
Preconditions:	The system is connected to a linear program solver and a description logic reasoner
Description:	The system checks the PTBox for consistency after it has checked the TBox consistency before. If both tests are successful, the result is returned to the application. [Exception: Computation of the consistency fails.]
Exceptions:	*Computation of the consistency fails*: The application is told which part of the consistency check failed or if it failed for connectivity reasons with solver or reasoner.
Postconditions:	none

Table 4.4: Compute Probabilistic Terminology Consistency

Non-functional Requirements

The non-functional requirements imposed on the intended system are presented in the following short list, bullet ☑ represents must-haves and bullet □ nice-to-haves.

1. Support RacerPro as description logic reasoner ☑(NF 1)
2. Support for further description logic reasoner□(NF 2)
3. Support for one linear program solver☑ (NF 3)
4. Support for further linear program solver □(NF 4)
5. Change linear program solver during runtime□(NF 5)
6. Support for multiple platforms ☑(NF 6)
7. Provide a basic graphical user interface□(NF 7)
8. Support for other probabilistic entailments□(NF 8)
9. Change entailment algorithm during runtime □(NF 9)

4.1.2 Description Logic Reasoner

A description logic reasoner is the kernel software providing reasoning services to knowledge-based applications using an interface (see figure 4.2). The main reasoning task regarding the TBox are satisfiability and subsumption of concepts. ABox reasoning services are checking consistency against a TBox and instance retrieval.

At the time of writing this thesis some of the reasoners have made the transition from research system to commercial product [10, 5, 6]. This indicates there is a market developing for these systems. An overview of description logic reasoners is available on the website of Ulrike Sattler [56]. Distinguishing features of the reasoners are the expressiveness of the underlying description logic, used inference algorithms, support for ABoxes and supported API's. For reasoners, which turned into commercial products, free research licences may be obtained.

In the following paragraphs a selection of description logic reasoners is presented in short. For detailed information please visit the reasoners websites and have a look at their manuals.

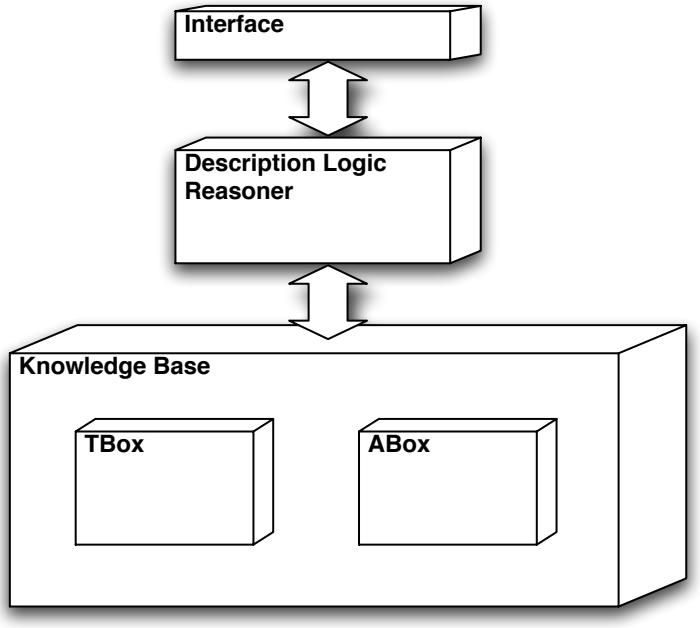


Figure 4.2: Description Logic Reasoner Architecture

RacerPro

The description logic reasoner RacerPro[1] is a product offered by Racer Systems GmbH & Co. KG [10]. This software has its roots in fundamental research by the University of Hamburg (Hamburg, Germany). Current research and development for Racer is undertaken by Ralf Möller at Technische Universität Hamburg-Harburg (Hamburg, Germany) and by Volker Haarslev at Concordia University (Quebec, Canada).

RacerPro is implemented using the Common Lisp programming language. Its supported underlying description logic is $\mathcal{ALCQHI}_{\mathcal{R}+}$ also known as $\mathcal{SHIQ}$. This reasoner provides reasoning services for several TBoxes and ABoxes. The implemented inference algorithm is based on a highly optimised tableau calculus.

In order to build a knowledge based system on top of RacerPro one may run it as a server and has the choice between the following interfaces:

DIG over HTTP, a Java API over TCP/IP or a Lisp API over TCP/IP.

It may also be setup as a batch system using file I/O.

Together with this reasoner a graphical user interface client, called Racer Porter, is delivered. This client supplies management and manipulation capabilities for one or more RacerPro Servers.

[1] RACER stands for **R**enamed **A**Box and **C**oncept **E**xpression **R**easoner.

RacerPro comes with a user's guide [51] and a reference handbook [50].

FaCT++

FaCT++ [3] description logic reasoner is the successor of the FaCT[2] system [30], developed by Ian Horrocks at the University of Manchester (England, United Kingdom). It is released under a GNU General Public License. Further research and development is carried out in Manchester by Ian Horroks and Dimitry Tsarkov.

The system is implemented in the C++ programming language, supports the underlying description logic $\mathcal{SROIQ}$ and uses a tableau based inference algorithm.

Knowledge based systems may access FaCT++ stand alone using a Lisp-like interface as well as a server using the DIG Interface over HTTP.

KAON2

The KAON2[3] [6] description logic reasoner is commercially available through ontoprise GmbH [9]. It has been developed by Boris Motik in order to validate the algorithms, he developed for his PhD Thesis at the Research Center for Information Technologies (FZI) (Karlsruhe, Baden-Württemberg, Germany). Contributing research efforts have been made at the University of Karlsruhe (Germany) and University of Manchester (United Kingdom).

The system is implemented in the Java programming language. As underlying description logic it supports $\mathcal{SHIQ}$. For the inference KAON2 uses algorithms which reduce the problem to a disjunctive datalog program.

A knowledge based application may access this reasoner using either RMI or the DIG interface.

4.1.3 Linear Program Solver

The first linear program solvers were provided by companies like IBM in order to solve their clients problems and sell their computers [46].

The file format developed for IBM's MPS[4] became the de facto standard for the description of linear programs [26]. This format, which was developed in a time when computers

[2]FaCT stands for **F**ast **C**lassification of **T**erminologies
[3]KAON stands for **Ka**rlsruhe **On**tology
[4]MPS stands for **M**athematical **P**rogramming **S**ystem

were programmed by punched cards, is still usable together with most linear program solvers existing today. IBM's MPSX-System, where X stands for extended and which had replaced MPS, was pulled off the market in 2005 [31].

An overview for available linear programming software may be found in the 2005 Linear Programming Software Survey [19].

In the following paragraphs a selection of linear program solvers is presented. For detailed information please visit the solvers websites and have a look at their manuals.

OR-Objects

OR-Objects is a Java framework for rapid development of applications in operations research, science and engineering, developed by DRA Systems [1]. It comes as a jar-file under a freeware licence.

The framework provides packages for mathematical programming, numerical problems, graphs and geometry. The package drasys.or.mp.lp contains a solver for linear programs using the simplex algorithm.

Knitro

Knitro [7] is a commercial software package designed to find solutions to numerous optimisation problems, one of these is linear programming. It is offered by Ziena Optimization Inc.

The software package provides APIs to C/C++, Fortran, Java and is available for several computing platforms. Optimal solutions are found using variants of the interior-point algorithm.

mosek

The mosek optimisation software[4] by MOSEK ApS is designed for large scale mathematical optimisation problems.

In order to interface with mosek one may use a MPS file interface or APIs for several programming languages (C, Fortran, Java, C#). Mosek implements both simplex and interior-point based algorithms for the solution of linear programs. The software suite is available for several computing platforms.

4.1.4 Related Systems

In 2005, Thomas Lukasiewicz made the nmproblog system[41, 38] available as a practical result of his research. The system computes entailment in variable strength nonmonotonic probabilistic logics (System P, System Z, lexicographic). The underlying logic is predicate logic.

nmproblog may be downloaded as an executable for the Linux operating system on x86 machines. It needs the linear programm solver lpsolve[8] be installed.

4.2 Architecture & Design

In this section the architecture and design of the prototype is described.

4.2.1 Architecture

The architecture for the ContraBovemRufum prototype (see figure 4.3) will extend the layered architecture, we have seen for description logic reasoner in figure 4.2. One may interpret the prototype as another layer on top. It provides the services, which are described in the Use Cases in subsection 4.1.1, through an interface.

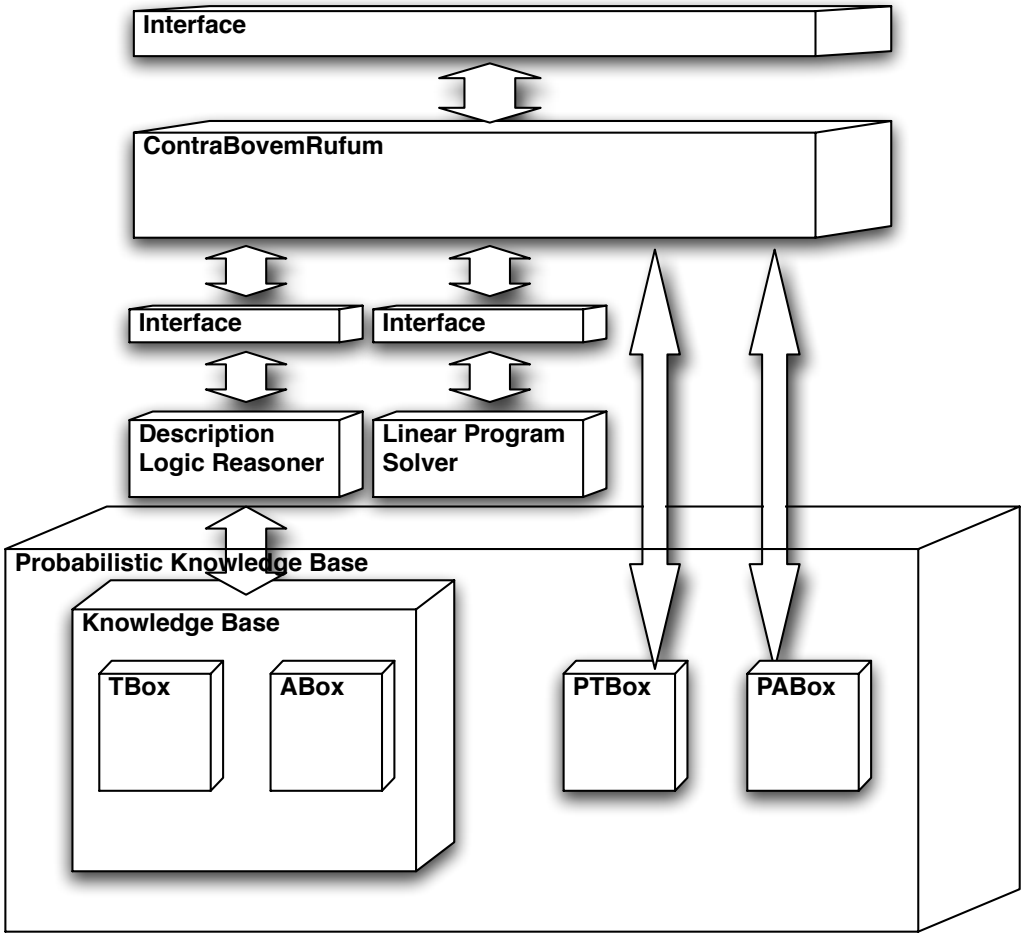


Figure 4.3: ContraBovemRufum Architecture

The bottom Knowledge Base Layer is expanded to a Probabilistic Knowledge Base by adding the PTBox as well as the PABox. Access to TBox and ABox remains exclusive to the description logic reasoner. If the prototype has to access the Probabilistic Knowledge Base it will do so via an interface to the description logic reasoner to its exclusive part or

directly to the probabilistic part.

In order to provide the reasoning services a linear program solver is placed in the intermediate layer in addition to the description logic reasoner. It will be used by the system through one of its interfaces.

All described reasoners in subsection 4.1.2 are equipped with a "remote" interface and run as servers; therefore it is possible to distribute this component over a network.

The examined solvers in subsection 4.1.3 all own interfaces which are linked as libraries into the application. If the solver should be run as a server connected over a network, more development effort is required or another solver, which provides such functionality, has to be chosen. Sending linear programs over a network is feasible due to the fact that they typically may be represented as sparse matrices [53].

Conclusive one can state that the proposed architecture is highly flexible and allows for easy distribution. With the possibility of distribution one can fall back onto an additional pool of performance if need be.

4.2.2 Design

The design of the core package (see figure 4.4) of ContraBovemRufum prototype has to solve the following difficult problem:

Achieve all functionality stated in the use cases while having the intended flexibility and extensibility in mind which derives especially from the non-functional requirements NF 2, NF 4, NF 5, NF 8 and NF 9.

The proposed solution rests upon the GoF Design Patterns[23] Abstract Factory and Builder.

One factory produces all the description logic reasoner related products. It is called *ReasonerFactory* and creates the two abstract products *Reasoner* and *ConditionalConstraint*. In order to satisfy NF 2, one has to create subclasses of the factory and its products and has to do minimal local amendments to the classes *ReasonerFactory* and *CorePreferences* to be able to choose the new reasoner by preference. Already implemented and tested *ReasonerFactories* and products remain untouched and therefore they do not have to be tested again and do not need to be recompiled.

Another factory produces the linear program solver related products. Naturally it is called *SolverFactory* and produces *Solver*. To satisfy NF4 and NF 5, an analogous approach as described above for the new reasoner is available.

The *ProbabilisticKnowledgeBase* implements both the *ProbabilisticKBInterface* and the *ProbabilisticEntailmentInterface*. Because of that it acts as a data container object holding the PTBox and PABox and references on TBox and ABox, a manager for its data and director for all operations on the *ProbabilisticEntailmentInterface*.

A PTBox is designed as a *SetOfConditionalConstraints* which is a normal set with the additional functionality of checking its own satisfiability with respect to a *ProbabilisticKnowledgeBase*.

A PABox represents a standard map containing names of individuals as keys and *SetOfConditionalConstraints* as values.

Computation of the entailment is carried out by the abstract builder class *TightEntailment* and its concrete children *TightLexEntailment* and *TightLogEntailment*. The build is initiated by the *ProbabilisticKnowledgeBase* as director. The employment of the builder pattern enables us to choose the entailment by preference at runtime (NF 9). Also further entailments may be integrated easily by subclassing *TightEntailment* (NF 8) and minimal amendmends to the director and *CorePreferences* class.

At this time we gained *TightLogEntailment* as further entailment for free since it is needed by *TightLexEntailment*. This is also the way to go if a new improved algorithm for an already existing entailment has to be implemented. The advantage is that an already tested and working algorithm needs not to be touched.

Overall the design facilitates the extensibility of the system into directions lined out in the nonfunctional requirements NF 2, NF 4 and NF 8. Further it allows for run time flexibility in the choice of algorithm and solver(NF 9, NF 5).

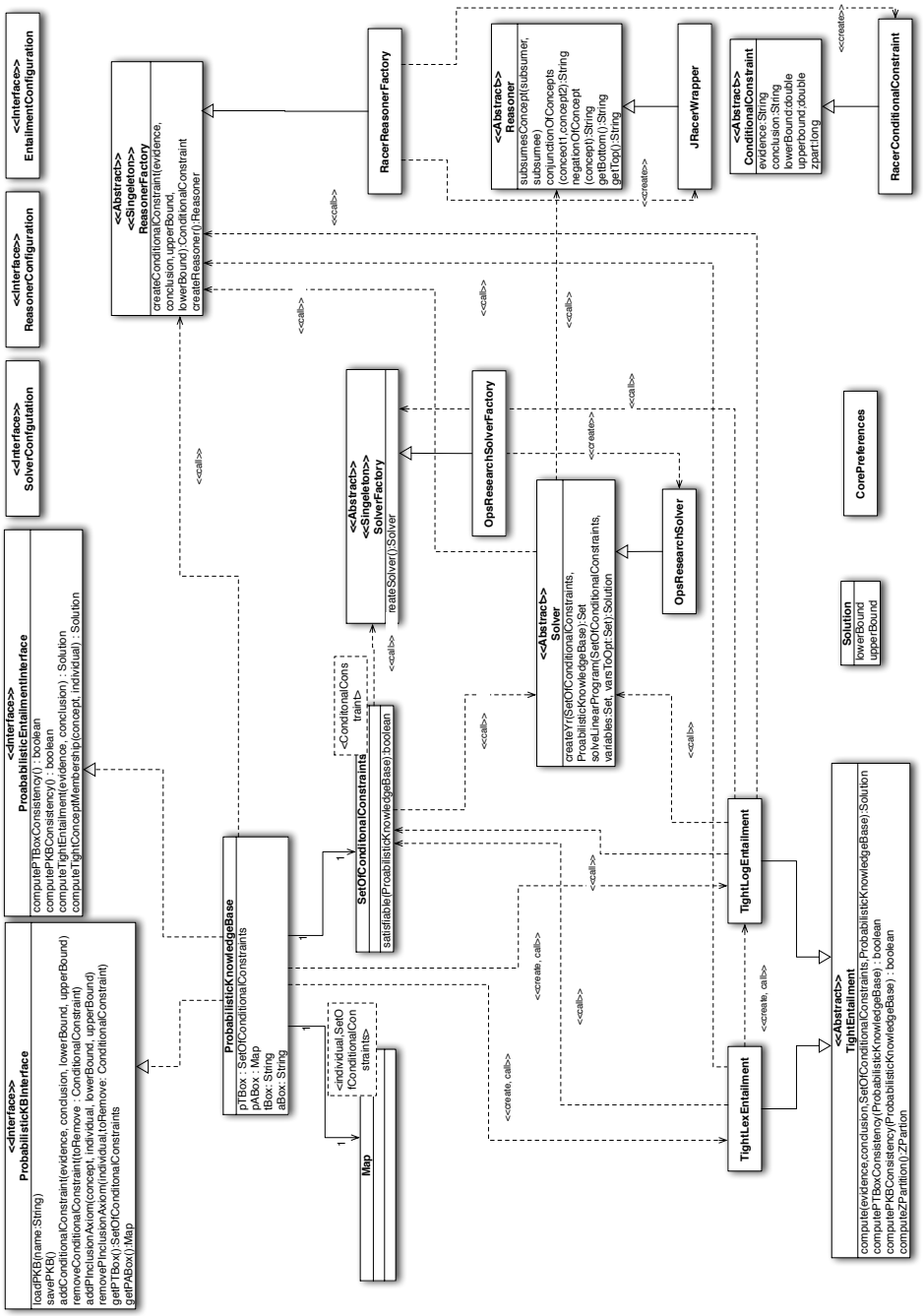


Figure 4.4: ContraBovemRufum Design

4.3 Implementation

Here some remarks regarding the prototypical implementation are stated.

4.3.1 Choice of programming language

Java™1.5 is the chosen implementation language for the ContraBovemRufum prototype. In the following text reasons for this decision are presented along some categories from [36].

With the nonfunctional requirement NF 6 in mind the choice of Java delivers a lot of advantages since a Java virtual machine is available for several hardware platforms and operation systems.

Also Java allows the use of natural API's provided by reasoner and solver. Further on this enables the use of powerful frameworks like the Java collection framework[59] and its extensions [57].

4.3.2 Choice of Description Logic Reasoner

Since the support of the RacerPro reasoner was a requirement, the development of the prototype started with this reasoner. The use of RacerPro turned out to be fairly easy due to its good documentation and JRacerAPI.

The prototype is capable to manage probabilistic knowledge bases, using the Racer language for concepts, and compute their entailment.

Support for further reasoners has not been implemented so far.

4.3.3 Choice of Linear Program Solver

As one can see in Linear Programming Software Survey [19] there are many linear programming software packages to choose from and there is no claim for being complete. The choice of Java as implementation language for the prototype provoked some solvers being more preferable for integrating in a Java environment.

The results of this search are the three solvers presented in subsection 4.1.3. For now the prototype uses the pure Java solver out of OR-Objects.

Support for further solvers has not been implemented so far.

4.3.4 Computation of the variables y_r

The way how the variables y_r are computed in table 3.2 is rather inefficient. Therefore the method computeYr() on *Solver* objects does the computation using a tree-structure with a depth-first search. Each node has three branches $D_i \sqcap C_i$, $\neg D_i \sqcap C_i$ or $\neg C_i$ were i is the node layer and the number of the selected conditional constraint. Branches of the tree that can not contain a variable are cut out.

In order to improve the performance, the search algorithm could be parallelised , but this requires that the description logic reasoner selected is capable of handling several requests simultaneously. At the moment RacerPro is not able to handle parallel requests; one could use the software RacerManager in combination with several RacerPro instances to achieve this behaviour .

Chapter 5

Average Case Analysis

After a successful implementation of the algorithm proposed in [24] (see appendix A for a copy of the algorithm) a further objective of this thesis was the analysis of its average case behaviour.

Already during the implementation of the class TightLexEntailment it became obvious, that this algorithm will perform in the average case like $O(2^n)$, where n is the number of conditional constraints within the ith part of the z-Partition.

In order to explain this conclusion, one has to take a closer look at the algorithm. The algorithm may be divided into three parts:

Part 1 (Line 1 - 2) tests, if the evidence concept is satisfiable against the TBox($\mathcal{T}_g$) and a set $\mathcal{F}$ of all ABox and PABox axioms bound to a certain individual. $\mathcal{F}$ is only relevant, if probabilistic concept membership is computed.

Part 2 (Line 3 - 13) computes lexicographic minimal sets of conditional constraints that are not in conflict with the verified evidence.

Part 3 (Line 14 - 19) computes the tightest entailed bounds using the previously computed sets and the conclusion concept.

The reason for poor average case performance is within the second part in line 7. Here the computation of the power set for each part $\mathcal{D}_i$ of the z-partition is required in order to iterate through all possible subsets $\mathcal{G}$. A power set has 2^n elements (see proof in [60]) and the algorithm visits all of them. Therefore the average complexity is determined to be $O(2^n)$ and it has been shown that the algorithm is intractable.

Here are some ideas on how to modify the algorithm to improve its average case performance:

As the first idea "lazy power set computation" is introduced. By this is meant, that one starts with $\mathcal{G}$ as the set containing all elements and then with all subsets, where we have $(n-1)$ elements and so on. The cardinality of these sets of subsets is given by $\binom{n}{n-i}$, where n is the number of elements and i the number of omitted elements.

If a set of subsets is found, where some sets are satisfiable, the process may be stopped since we are interested only in those kind of sets which satisfy as many conditional constraints as possible. Still for the worst case all subsets down to the level where the subsets only contain one element have to be computed. But on average the algorithm has to visit less subsets $\mathcal{G}$ than the implemented algorithm.

The second idea is to combine "lazy power set computation" with binary search. This works as follows:

We start with the set of subsets containing $\binom{n}{n-\lceil \frac{n}{2} \rceil}$ and test its elements for satisfiability. If at least one set is satisfied, the search continues within the upper half between n and $\lceil \frac{n}{2} \rceil$; otherwise the search continues in the lower half in the same manner.

These two ideas have been already applied to the algorithm in [39].

A third idea is to introduce some heuristic method before performing binary search in order to reduce the search space. For example we could pick a random subset $\mathcal{G}$ of $\mathcal{D}_j$ and test it for satisfiability. If $\mathcal{G}$ is indeed satsfiable the cardinality of $\mathcal{G}$ is set as new lower bound for the binary search. This makes sense because computing the satisfiability of a random subset is relativly cheap compared to computing set of subsets with $n-i$ elements.

All ideas presented are definitely fields of interest for future research. The ContraBovem-Rufum Prototype is prepared for easy integration of these ideas (see subsection 4.2.2).

Chapter 6

Summary

This chapter will provide a summary of the thesis, give final conclusions and recommendations for future research and development topics.

6.1 Summary & Conclusion

A prototypical implementation of the inference algorithm for probabilistic description logics proposed by Giugno and Lukasiewicz in [24] has been realised. Afterwards the average case behaviour of the implemented algorithm was evaluated analytically.

6.1.1 Prototype

The ContraBovemRufum Prototype may be seen as a basic software framework that is capable of easily integrating further description logic reasoners , linear program solver and probabilistic entailment algorithms. Within the scope of this thesis the description logic reasoner RacerPro was integrated together with the linear program solver from OpsResearch in order to provide the new probabilistic reasoning services *probabilistic subsumption* and *probabilistic instance checking*.

For determining the probabilistic lexicographic entailment, papers [25, 24] have been discussed and the proposed algorithm has been implemented.

As a freebie the prototype is able to handle tight probabilistic logical entailment. This entailment does not allow exceptions in the Probabilistic Knowledge Base.

6.1.2 Average case analysis

The average case analysis concluded analytically that the implemented algorithm is intractable. But that does not mean that there is no room for improvement of the average case behaviour, since the algorithm uses a brute force approach to compute the lexicographic minimal set.

6.2 Recommendations

Here some ideas for further research and development of the implemented ContraBovemRufum System are given:

- Integration of further solvers, description logic reasoners and of general description logic interfaces DIG and DIG 2.0, enabling a broader application of the system and selection of the user-preferred subsystems.
- Research and development of new tractable algorithms by applying different search strategies to determine lexicographic minimal sets.
- Development of an analysis package to evaluate new algorithms.
- Optimisation of the new algorithms by threaded computation of the variables y_r, of the parts of the z-partition and of lexicographic minimal sets.
- Improvement of the management functionality for Probabilistic Knowledge Bases by implementation of a PKB Broker, which allows to save a PKB using XML and convert it from one reasoner language to another.
- Research and development of applications which can be realised on top of the ContraBovemRufum System

Bibliography

[1] Or-objects, 2000. This is an electronic document retrieved from http://opsresearch.com/index.cgi . Date retrieved: January 18, 2007.

[2] George Dantzig Memorial Site. Institute of Operations Research and Management Science, 2005. This is an electronic document retrieved from http://www2.informs.org/History/dantzig/index.htm. Date retrieved: January 22, 2007.

[3] Fact++, 2006. This is an electronic document retrieved from http://owl.man.ac.uk/factplusplus/. Date retrieved: January 18, 2007. Date last modified: December 19, 2006.

[4] mosek. MOSEK ApS, 2006. This is an electronic document retrieved from http://www.mosek.com/index.html. Date retrieved: January 23, 2007.

[5] Pellet, 2006. This is an electronic document retrieved from http://pellet.owldl.com/. Date retrieved: January 18, 2007. Date last modified: November 7, 2006.

[6] Kaon2, 2007. This is an electronic document retrieved from http://kaon2.semanticweb.org/. Date retrieved: January 18, 2007.

[7] Knitro. Ziena Optimization Inc., 2007. This is an electronic document retrieved from http://www.ziena.com/knitro.htm. Date retrieved: January 23, 2007.

[8] lpsolve, 2007. This is an electronic document retrieved from http://sourceforge.net/projects/lpsolve/. Date retrieved: January 27, 2007.

[9] ontoprise, 2007. This is an electronic document retrieved from http://www.ontoprise.de/content/. Date retrieved: January 18, 2007.

[10] Racer systems, 2007. This is an electronic document retrieved from http://www.racer-systems.com/index.phtml. Date retrieved: January 18, 2007. Date last modified: January 17, 2007.

[11] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[12] Fahiem Bacchus. *Representing and reasoning with probabilistic knowledge: a logical approach to probabilities*. MIT Press, Cambridge, MA, USA, 1990.

[13] J. E. Beasley, editor. *Advances in linear and integer programming*. Oxford University Press, Inc., New York, NY, USA, 1996.

[14] Norman L. Biggs. *Discrete Mathematics*. Oxford University Press, Oxford, revised edition edition, 1989.

[15] Rachel A. Bourne and Simon Parsons. Connecting lexicographic with maximum entropy entailment. In *ESCQARU*, pages 80–91, 1999.

[16] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. John Wiley & Sons, Inc., New York, 1951.

[17] Zhongli Ding and Yun Peng. A probabilistic extension to ontology language owl. *hicss*, 04:40111a, 2004.

[18] Friedrich Esser. *Java 2 - Designmuster und Zertifizierungswissen*. Galileo Computing, 2001.

[19] Robert Fourer. Linear programming software survey. *OR/MS Today*, 2005.

[20] Robert Fourer, David M. Gay, and Brian W. Kerninghan. *AMPL - A Modeling Language for Mathematical Programming*. Thomson Books/Cole, 2003.

[21] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*. Addison-Wesley Professional, September 2003.

[22] Christine Froidevaux and Daniel Kaiser. *Non-Standard Logics for Automated Reasoning*, chapter 7, pages 179–212. Academic Press Limited, 1988.

[23] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[24] Rosalba Giugno and Thomas Lukasiewicz. P-$\mathcal{SHOQ}(\mathbf{D})$: A probabilistic extension of $\mathcal{SHOQ}(\mathbf{D})$ for probabilistic ontologies in the semantic web. Technical Report INFSYS RR-1843-02-06, TU Wien, 2002.

[25] Rosalba Giugno and Thomas Lukasiewicz. P-$\mathcal{SHOQ}(\mathbf{D})$: A probabilistic extension of $\mathcal{SHOQ}(\mathbf{D})$ for probabilistic ontologies in the semantic web. In *JELIA*, pages 86–97, 2002.

[26] Dr. Ralf Gollmer. Optimierungsverfahren und ihre Umsetzung in Optimierungssoftware 1 - lineare Technical report, Universität Duisburg-Essen, 2005.

[27] Rolf Haenni. Towards a unifying theory of logical and probabilistic reasoning. In *ISIPTA*, pages 193–202, 2005.

[28] Theodore Hailperin. *Boole's Logic and Probability.* Elsevier Science Publishers B.V., second edition edition, 1986.

[29] I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[30] Ian Horrocks. Fact, 2006. This is an electronic document retrieved form `http://www.cs.man.ac.uk/~horrocks/FaCT/`. Date retrieved: January 19, 2007. Date last modified: April 29, 2003.

[31] IBM. Software withdrawal: Ibm mathematical programming system extended/370 v2.2, February 2005. This is an electronic document retrieved from `http://www-306.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=an&subtype=ca&appname=Demonstration&htmlfid=897/ENUS905-026`. Date retrieved: January 18, 2007. Date last modified: November 7, 2006.

[32] Manfred Jaeger. Probabilistic reasoning in terminological logics. In *KR*, pages 305–316, 1994.

[33] Daphne Koller, Alon Y. Levy, and Avi Pfeffer. P-classic: A tractable probabilistic description logic. In *AAAI/IAAI*, pages 390–397, 1997.

[34] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[35] Daniel Lehmann. Another perspective on default reasoning. *Annals of Mathematics and Artificial Intelligence*, 15:61–82, 1995.

[36] Dennis Ludwig. Language considerations. *CrossTalk-The Journal of Defense Software Engineering*, Febuary 2003.

[37] T. Lukasiewicz. Probabilistic logic programming under inheritance with overriding, 2001.

[38] T. Lukasiewicz. Nonmonotonic probabilistic logics under variable-strength inheritance with overriding: Algorithms and implementation in nmproblog. In *4th International Symposium on Imprecise Probabilities and Their Applications, Pittsburgh, Pennsylvania*, 2005.

[39] T. Lukasiewicz. Probabilistic description logics for the semantic web. Technical Report INFSYS RR-1843-06-05, TU Wien, 2006.

[40] Thomas Lukasiewicz. Probabilistic default reasoning with conditional constraints. *CoRR*, cs.AI/0003023, 2000.

[41] Thomas Lukasiewicz. Nmproblog version 1.0, 2005. This is an electronic document retrieved from `http://www.kr.tuwien.ac.at/staff/lukasiew/nmproblog.html`. Date retrieved: January 27, 2007.

[42] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artif. Intell.*, 13(1-2):27–39, 1980.

[43] Drew V. McDermott and Jon Doyle. Non-monotonic logic i. *Artif. Intell.*, 13(1-2):41–72, 1980.

[44] Frank Mittelbach and Michel Goossens. *Der LaTeX Begleiter*. Pearson Studium, 2005.

[45] Gerhard Paass. *Non-Standard Logics for Automated Reasoning*, chapter 8, pages 213–251. Academic Press Limited, 1988.

[46] Manfred Padberg. *Linear Optimization and Extensions*. Springer, 1995.

[47] Scott Pakin. The comprehensive latex symbol list, September 2005.

[48] Judea Pearl. *Probabilistic Resoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.

[49] Judea Pearl. System z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 121–135, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[50] Racer Systems GmbH & Co. KG. *RacerPro Reference Manual*, 2005. Version 1.9.

[51] Racer Systems GmbH & Co. KG. *RacerPro User's Guide*, 2005. Version 1.9.

[52] Raymond Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[53] Brian W. Kernighan Robert Fourer, David M. Gay. A modeling language for mathematical programming. *Management Science*, 36:519–554, 1990.

[54] James Rumbaugh. Getting started: Using use cases to capture requirements. *Journal of Object-Oriented Programming*, 23:8–12, September 1994.

[55] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[56] Ulrike Sattler. Description logic reasoners, 2007. This is an electronic document retrieved from `http://www.cs.man.ac.uk/~sattler/reasoners.html`. Date retrieved: January 18, 2007.

[57] Sergej Schütz Siegfried Hodri. Mathcollection - release 1.3.2, November 2005. This is an electronic document retrieved from `http://www.iis.uni-stuttgart.de/personen/lippold/MathCollection/index-de.html`. Date retrieved January 30, 2007.

[58] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.

[59] Sun. Java collection framework. This is an electronic document retrieved from `http://java.sun.com/j2se/1.5.0/docs/guide/collections/`. Date retrieved: January 30, 2007.

[60] H. F. Mattson, Jr. *Discrete Mathematics with Applications*, chapter 3, pages 120–121. John Wiley & Sons, Inc., 193.

[61] Christian Ullenboom. *Java ist auch eine Insel - Programmieren mit der Java Standard Edition Version 6.* Galileo Computing, 6 edition, 2007.

Appendix A

Algorithm

This algorithm was proposed in [24] and has been implemented in the ContraBovemRufum prototype.

Input: Consistent generic probabilistic terminology $\mathcal{P}_g$, set of terminological axioms and conditional constraints $\mathcal{F}$, and two concepts C and D .
Output: Pair of real numbers $(l, u) \in [0, 1]$ such that $\mathcal{F} \mid\!\sim_{tight} (D|C)[l, u]$ under $\mathcal{P}_g$.
Notation: $(\mathcal{D}_0, \ldots, \mathcal{D}_k)$ denotes the z-partition of $\mathcal{P}_g$.

1. $\mathcal{R} := \mathcal{T}_g \cup \mathcal{F} \cup \{(C|\top)[1, 1]\}$;
2. **if** $\mathcal{R}$ is unsatisfiable **then return**$(1, 0)$;
3. $\overline{\mathcal{H}} := \{\emptyset\}$;
4. **for** $j := k$ **downto** 0 **do**
5. $\quad n := 0$;
6. $\quad \overline{\mathcal{H}}' := \emptyset$;
7. $\quad$ **for each** $\mathcal{G} \subseteq \mathcal{D}_j$ and $\mathcal{H} \in \overline{\mathcal{H}}$ **do**
8. $\quad\quad$ **if** $\mathcal{R} \cup \mathcal{G} \cup \mathcal{H}$ is satisfiable **then**
9. $\quad\quad\quad$ **if** $n = |\mathcal{G}|$ **then** $\overline{\mathcal{H}}' := \overline{\mathcal{H}}' \cup \{\mathcal{G} \cup \mathcal{H}\}$;
10. $\quad\quad\quad$ **else if** $n < |\mathcal{G}|$ **then** $\overline{\mathcal{H}}' := \{\mathcal{G} \cup \mathcal{H}\}; \quad n := |\mathcal{G}|$;
11. $\quad$ **end**
12. $\overline{\mathcal{H}} := \overline{\mathcal{H}}'$
13. **end**
14. $(l, u) := (1, 0)$
15. **for each** $\mathcal{H} \in \overline{\mathcal{H}}$ do
16. $\quad$ compute $c, d \in [0, 1]$ such that $\mathcal{R} \cup \mathcal{H} \models_{tight} (D|\top)[c, d]$
17. $\quad (l, u) := (min(l, c), max(u, d))$
18. **end**
19. **return** (l, u)

Appendix B

Nomenclature

B.1 Logical Symbols

a, b, c	primitive propositions
ϕ, ψ	propositional formulae
$\neg\phi$	negation or not
$\phi \wedge \psi$	conjunction or and
$\phi \vee \psi$	disjunction or or
$\models$	entails, satisfies, models
$\models_{lex}$	lexicographic entails
$(\phi : \psi)$	default
D	set of defaults
D_i	ith part of the z-partition of $\mathcal{D}$

B.2 Description Logic Identifiers

Identifiers used to indicate the expressiveness of description logics.

$\mathcal{AL}$	attributive language, containing atomic concept, universal concept, bottom concept, atomic negation, intersection, value restriction and limited existential quantification
$\mathcal{C}$	negation
$\mathcal{R}^+$	transitive roles
$\mathcal{I}$	inverse roles

$\mathcal{N}$	number restriction
$\mathcal{Q}$	qualified number restriction
$\mathcal{O}$	nominals
$\mathcal{H}$	role hierarchies
$\mathcal{S}$	short for $\mathcal{ALCR}^+$

B.3 Symbols of Description Logic

B.3.1 Description Logic $\mathcal{AL}$

C, D	concepts
R, S	roles
A	atomic concept
$\top$	universal concept
$\bot$	bottom concept
$\neg A$	atomic negation
$C \sqcap D$	intersection
$\forall R.C$	value restriction
$\exists R.\top$	limited existential quantification
$C \sqsubseteq D$	inclusion axiom
$C \equiv D$	equality axiom
$\mathcal{I}$	interpretation
$\Delta^{\mathcal{I}}$	domain of interpretation
$\cdot$	interpretation function

B.3.2 Further constructors $\mathcal{C}, \mathcal{R}^+, \mathcal{I}, \mathcal{N}, \mathcal{Q}, \mathcal{O}$ and $\mathcal{H}$

$\neg C$	negation
$C \sqcup D$	union
$\exists R.C$	full existential quantification
R^+	transitive role
R^-	inverse role
$\leq nR$ or $\geq nR$	number restriction
$\leq nR.C$ or $\geq nR.C$	qualified number restriction

$\{o\}$	nominals
$R \sqsubseteq S$	inclusion axioms (role hierarchies)

B.3.3 Probabilistic lexicographic extension

l	lower bound
u	upper bound
$(C\|D)[l,u]$	conditional constraint

B.4 Math Terms

B.4.1 Partition of a Set

"A **partition** of a set X is a family $\{X_i | i \in I\}$ of non-empty subsets of X such that

(i) X is the union of the sets X_i $(i \in I)$,

(ii) each pair X_i, X_j $(i \neq j)$ is disjoint.

The subsets X_i are called the **parts** of the partition."[14]

Appendix C

Tools & Support

This appendix lists all support and tools used.

C.1 Support

The stories 1.1 and 1.2 have been illustrated by Christina Wenterodt. This thesis has been proof read by Volker G. Näth for typing errors. The graphical user interface has been programmed by Martin Malzahn.

C.2 Tools

The thesis was written using an Apple Powerbook G4 @ 1.25Ghz, 1.25GB running OS X 10.4.8. It has been typeset using teTex 7.9.0 and TexShop 2. The bibliography was managed by JabRef 2.1. As references for LaTeX[44] and [47] have been used.

All graphics have been created with OmniGraffle Pro 3.2.4.

The ContraBovemRufum system was developed in Java 1.5 with Eclipse 3.2 on the same system. As references for Java [61] and [18] have been used. All tests have been made with JUnit 3.8. The system uses the OpResearch solver [1] and MathCollection [57]. RacerPro Version 1.9.0 has been used as description logic reasoner [10].

Appendix D

Winged Penguins Example

D.1 TBox

```
(in-tbox wingedpenguins)
(implies penguin bird)
```

D.2 Linear program concept bird verified

```
var Ybpwf >=0;
var YbpWf >=0;
var YbpwF>=0;
var YbpWF>=0;
var YbPwf>=0;
var YbPWf>=0;
var YbPwF >=0;
var YbPWF>=0;
var YBP>=0;

maximize cost:
Ybpwf + YbpWf + YbpwF+ YbpWF + YbPwf + YbPWf + YbPwF + YbPWF + YBP;


subject tolWB:
0.05*Ybpwf + (-0.95)*YbpWf+ 0.05*YbpwF+ (-0.95)*YbpWF + 0.05*YbPwf +
    (-0.95)*YbPWf + 0.05*YbPwF + (-0.95)*YbPWF + 0*YBP >= 0;

subject to lFB:
```

```
0.1*Ybpwf + 0.1*YbpWf + (-0.9)*YbpwF+ (-0.9)*YbpWF+ 0.1*YbPwf+ 0.1*
    YbPWf+ (-0.9)*YbPwF + (-0.9)*YbPWF + 0*YBP >= 0;

subject tolFP:
1*Ybpwf + 1*YbpWf + 0*YbpwF+ 0*YbpWF+ 0*YbPwf + 0*YbPWf + 0*YbPwF +
    0*YbPWF + 0*YBP>= 0;

subject tolBT:
0*Ybpwf + 0*YbpWf + 0*YbpwF+ 0*YbpWF+ 0*YbPwf + 0*YbPWf + 0*YbPwF +
    0*YbPWF + (-1)*YBP>= 0;

subject touWB:
0*Ybpwf + 1*YbpWf+ 0*YbpwF+ 1*YbpWF + 0*YbPwf + 1*YbPWf + 0*YbPwF +
    1*YbPWF + 0*YBP >= 0;

subject to uFB:
(-0.05)*Ybpwf+ (-0.05)*YbpWf + 0.95*YbpwF+ 0.95*YbpWF+ (-0.05)*YbPwf+
     (-0.05)*YbPWf+ 0.95*YbPwF + 0.95*YbPWF + 0*YBP >= 0;

subject touFP:
(-0.95)*Ybpwf + (-0.95)*YbpWf + 0.05*YbpwF+ 0.05*YbpWF+ 0*YbPwf + 0*
    YbPWf + 0*YbPwF + 0*YbPWF + 0*YBP>= 0;

subject touBT:
0*Ybpwf + 0*YbpWf + 0*YbpwF+ 0*YbpWF+ 0*YbPwf + 0*YbPWf + 0*YbPwF +
    0*YbPWF + 1*YBP>= 0;

subject to One:
Ybpwf + YbpWf + YbpwF+ YbpWF + YbPwf + YbPWf + YbPwF + YbPWF + YBP =
    1;
```

D.3 Linear program concept penguin verified

```
var Ybpwf>=0;
var YbpWf>=0;
var YbpwF>=0;
var YbpWF>=0;
var YbPwf>=0;
var YbPWf>=0;
```

```
8  var YbPwF>=0;
   var YbPWF>=0;
10 var YBP>=0;

12 maximize cost:
   Ybpwf + YbpWf + YbpwF+ YbpWF + YbPwf + YbPWf + YbPwF + YbPWF + YBP;
14


16 subject tolWB:
   0.05*Ybpwf + (-0.95)*YbpWf+ 0.05*YbpwF+ (-0.95)*YbpWF + 0.05*YbPwf +
       (-0.95)*YbPWf + 0.05*YbPwF + (-0.95)*YbPWF + 0*YBP >= 0;
18
   subject to lFB:
20 0.1*Ybpwf + 0.1*YbpWf + (-0.9)*YbpwF+ (-0.9)*YbpWF+ 0.1*YbPwf+ 0.1*
       YbPWf+ (-0.9)*YbPwF + (-0.9)*YbPWF + 0*YBP >= 0;

22 subject tolFP:
   1*Ybpwf + 1*YbpWf + 0*YbpwF+ 0*YbpWF+ 0*YbPwf + 0*YbPWf + 0*YbPwF +
       0*YbPWF + 0*YBP>= 0;
24
   subject tolPT:
26 0*Ybpwf + 0*YbpWf + 0*YbpwF+ 0*YbpWF+ (-1)*YbPwf + (-1)*YbPWf + (-1)*
       YbPwF + (-1)*YbPWF + (-1)*YBP>= 0;

28 subject touWB:
   0*Ybpwf + 1*YbpWf+ 0*YbpwF+ 1*YbpWF + 0*YbPwf + 1*YbPWf + 0*YbPwF +
       1*YbPWF + 0*YBP >= 0;
30
   subject to uFB:
32 (-0.05)*Ybpwf+ (-0.05)*YbpWf + 0.95*YbpwF+ 0.95*YbpWF+ (-0.05)*YbPwf+
        (-0.05)*YbPWf+ 0.95*YbPwF + 0.95*YbPWF + 0*YBP >= 0;

34 subject touFP:
   (-0.95)*Ybpwf + (-0.95)*YbpWf + 0.05*YbpwF+ 0.05*YbpWF+ 0*YbPwf + 0*
       YbPWf + 0*YbPwF + 0*YbPWF + 0*YBP>= 0;
36
   subject touPT:
38 0*Ybpwf + 0*YbpWf + 0*YbpwF+ 0*YbpWF+ 1*YbPwf + 1*YbPWf + 1*YbPwF +
       1*YbPWF + 1*YBP>= 0;
```

```
40
   subject to One:
42 Ybpwf + YbpWf + YbpwF+ YbpWF + YbPwf + YbPWf + YbPwF + YbPWF + YBP =
       1;
```

Appendix E

CDROM *ContraBovemRufum*