University of Hagen

Computation of ranking functions for knowledge bases with relational conditionals

Master Thesis

Tobias Falke

Degree:	Master of Science Practical Computer Science
Supervisor:	Prof. Dr. Christoph Beierle Faculty of Mathematics and Computer Science Chair of Knowledge-Based Systems
Date of submission:	April 19, 2015

Abstract

In the field of knowledge representation and reasoning, conditionals provide a convenient way to express qualitative default rules of the type "If A, then usually B". Different systems have been suggested that allow default reasoning based on such conditionals, with system Z by Pearl being one of the most popular ones. Recently, Kern-Isberner and Beierle published an extension of system Z to a restricted first-order language. With this approach, a ranking function can be computed that is a model of the knowledge base. The necessary input is a partition of the conditionals and the domain of a first-order knowledge base, called a tolerance-pair, that satisfies certain tolerance-conditions.

In this thesis, the new system proposed by Kern-Isberner and Beierle is studied and analyzed. First, an implementation is provided that can be used to perform the necessary computations and investigate the resulting ranking functions. It is designed to be easily usable for such a purpose and is delivered together with this thesis. Further, the results obtained by evaluating the system with the implementation are presented. It turns out that for a first-order knowledge base, none, one or multiple tolerance-pairs exist, which depends on the expressed knowledge and the size of the domain. While the domain has to contain enough individuals to provide a representative for each conditional, adding unspecified individuals increases the number of tolerance-pairs. Three algorithms, following a generate-and-test and a backtracking-search approach, are presented that can be used to find tolerance-pairs. They are also part of the implementation.

Similar to system Z, a notion of minimality is introduced to identify preferred tolerance-pairs that induce minimal ranking functions for a first-order knowledge base. However, several minimal tolerance-pairs can exist and even a minimal tolerance-pair does not necessarily lead to a globally minimal ranking function as it is the case for system Z. The consequence relation provided by such a ranking function can handle all common benchmark problems as system Z does and is a preferential consequence relation. In addition, the system allows to model knowledge bases with the greater expressiveness of a first-order language and provides reasonable semantics for quantifications and free variables. Still open questions are whether a unique ranking function can be identified in this approach and how the inconsistency of a knowledge base can be proved. All in all, the new system turns out be a powerful system that seems to be an important step forward in order to pursue the goal of modeling human reasoning in a formal system.

Acknowledgements

I would like to thank my advisor, Prof. Dr. Christoph Beierle, for his patient guidance, feedback and support during the work on this thesis. I am also grateful to Nico Potyka for giving me an extensive introduction into the *Log4KR* library that I used for the implementation part of this thesis.

Contents

Lists

List of Definitions

List of Algorithms

1 Introduction

Since the first computers have been created in the middle of the last century, it was always obvious that there are certain tasks at which they perform significantly better than any human being. When it comes to mathematical calculations for example, there is no doubt a computer program can do them faster and more correctly. However, there are also a lot of tasks at which computer perform worse than humans or of which they are not capable at all. As long as computers have been around, researchers tried to find ways to change this fact and to enable computers to cope with more and more tasks as good as humans do – or even better. Within the computer science research community, these efforts gave rise to the field of artificial intelligence, in which methods and techniques to imitate human intelligence in computer programs are explored.

Recent advances show the tremendous opportunities that come with progress in this field. In 2011, IBM's computer system Watson won against human competitors in the quiz show Jeopardy, utilizing state of the art artificial intelligence techniques. Today, the system is used to improve lung cancer treatment by giving advice and decision support to medical experts in hospitals.[1] In 2012, Google announced that their self-driving cars completed 300.000 miles of autonomous driving without a single accident.[2] Introducing this technology to commercially available vehicles in the future will probably reduce the numbers of accidents and traffic deaths by a remarkable amount. The same year, researchers at Google also published that they were able to let a computer recognize cat faces in a set of millions of randomly chosen Youtube videos, without having introduced the concept of a cat to the program before.[3] These methods can help to find relevant information in huge datasets, a task at which humans perform poorly, facing the incredibly huge amount of information that is produced in the age of the internet.

If we want a computer to act as intelligent as a human being, one central question is what the crucial features of human intelligence are. An important one is the way we work with knowledge. If we look out of the window in the morning and see that the lawn is wet, we conclude that it must have rained last night. From the fact "the lawn is wet", applying a general rule that we learned by experience, the new fact "it has rained last night" is derived. If we then get to know the fact that the lawn is watered automatically every

[1] Bruce Upbin, Forbes.com, `http://www.forbes.com/sites/bruceupbin/2013/02/08/ibms-watson-gets-its-first-piece-of-business-in-healthcare/`, accessed on March 2nd, 2015

[2] Chris Urmson, Official Google Blog, `http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html`, accessed on March 2nd, 2015

[3] John Markoff, The New York Times, `http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html`, accessed on March 2nd, 2015

morning at 6 am, we retract our previous conclusion due to having a better explanation. However, unless we have a reason for not concluding "it has rained last night", it seems to be reasonable to make this conclusion. This flexible way of working with knowledge and deriving and retracting new knowledge from it is a fundamental property of human reasoning. It allows us to work with incomplete information and to always come to the best conclusions possible in a certain situation.

In artificial intelligence, the area of knowledge representation and reasoning is dedicated to model human reasoning as described in the example. Researchers develop formalisms that can be used to represent knowledge and to work with it similar to the way we do. These formalisms are then used as the core of applications that incorporate intelligent reasoning. IBM's Watson and Google's self-driving car are just some of the most popular examples of such applications.

A lot of different approaches for knowledge representation and reasoning have been proposed in the last decades. The basic and most simple system is that of propositional logic, in which knowledge can be represented by formulas and logical deduction provides a way to model reasoning. However, classical logical systems are too strong to model human reasoning, as they do not allow the withdrawal of a previous conclusion in the presence of new information. As a result, systems allowing so called non-monotonic reasoning have been developed, modeling the nature of human reasoning more accurately. Among different systems proposed for non-monotonic reasoning, system Z by Pearl (cf. [Pea90]) is one of the most popular ones. Its reasoning behavior shows a lot of the expected properties and all necessary information needed to make conclusions is solely derived from the existing knowledge. One of its limitations though is the base language which is propositional logic. In [KIB15], Kern-Isberner and Beierle propose an extension to first-order logic that allows for greater expressiveness.

The first goal of this thesis is to provide an implementation of the new system developed by Kern-Isberner and Beierle. The implementation is then used to compare it to its ancestor system Z and to point out the differences and similarities. Based on the implementation, the second goal is to evaluate the approach and to identify the benefits, issues and open questions. It is analyzed whether the proposed system can be used to more accurately model human reasoning in a formal system than with former approaches.

In Chapter 2, different approaches for non-monotonic reasoning based on conditionals are presented, providing the necessary background. Especially system Z, which is the basis for the new approach analyzed in this thesis, is discussed. The system proposed in [KIB15] is introduced in Chapter 3 and illustrated by an example. During the rest of the thesis, this system is analyzed and discussed. In Chapter 4, the created implementation is shown. Results of the evaluation of the system are presented in Chapter 5, representing the main part of the thesis. Finally, in Chapter 6, the main findings are summarized and open questions for further work on this topic are pointed out.

2 Background

This chapter provides the background for the content of the thesis. It introduces the field of knowledge representation and reasoning as well as the central terms *consequence relation* and *conditional*. In the main section different approaches for default reasoning that have been suggested within the research community are presented. Among them, especially system Z, which is the basis for the new approach analyzed in this thesis, is illustrated. Finally, issues that arise when moving to a first-order setting as well as different techniques for evaluating the quality of a system for default reasoning are discussed.

2.1 Knowledge representation and reasoning

Within the field of artificial intelligence, *knowledge* is one of the key concepts. If we want a computer program to be intelligent, it has to know as much as possible about its domain. This includes, for instance, knowing all objects that are relevant, their relationships, general rules that apply to them and the possible ways they can change. In order to make this knowledge usable for a computer program, it has to be encoded in a well-defined language. A precise definition of a *representation* always consists of two parts, as it is pointed out in Bench-Capon's definition:

> "Representation: [...] a set of syntactic and semantic conventions that makes it possible to describe things." [Ben90, p. 11]

The syntax is a set of rules defining how a given set of symbols can be combined to form the sentences of the representation language. The semantics define how these sentences should be interpreted and therefore associate a meaning to them. Both given, arbitrary sentences can be made that can express, for example, general rules of behavior or facts describing a certain situation. Obviously, having a well-defined representation of knowledge makes it possible to decouple the knowledge itself from the application that should use them. Although intelligent behavior could also be implemented explicitly in a computer program, independent representations of knowledge are more flexible and re-usable.

An important decision for every knowledge engineer is which representation to choose. While the human language allows us to express almost everything, there is no precise definition of its syntax and semantic. Two languages that are precisely defined are propositional and first-order logic, and they are the foundation of almost every formalism of knowledge presentation. In [BKI08], a general framework is introduced to specify log-

ical languages. Using this framework, we can give a more precise characterization for logic-based formalisms of knowledge representation:

> A logic-based knowledge representation language consists of a signature Σ, a set of formulas $\mathcal{L}_\Sigma$ built from the signature, a set of interpretations $Int(\Sigma)$ and a satisfaction relation $\models_\Sigma$ that connects formulas and interpretations, indicating whether a formula is valid in a given interpretation.

Although there are formalisms for knowledge representation that do not built upon classical logic, this characterization is acceptable in the context of this thesis, in which all discussed approaches are indeed logic-based. A central trade-off that we face when comparing different languages is the one between expressiveness and computational tractability, which also arises for the two basic logical languages. While the language should be capable of expressing the necessary knowledge, we have to make sure it can be handled by a computer program in a reasonable way.

The second concern of knowledge representation and reasoning is, as indicated by the name, *reasoning*. Given a set of existing knowledge, a knowledge base, the question is how additional knowledge can be derived from it. Once a representation of knowledge is defined, there has to be a mechanism, called inference, that derives conclusions in a way similar to human reasoning. In a classical logical system, logical deduction provides this kind of mechanism. From a formula A, a formula B can be deduced if B is valid in at least all of the interpretations in which A is valid. However, this leads to the set of knowledge growing monotonically, making it impossible to retract previously deduced formulas from the knowledge base when new knowledge is added.

As it has already been illustrated in the introduction, the withdrawal of previous conclusions is a central feature of human reasoning, helping us to cope with situations in which we have incomplete information and need to make reasonable assumptions. If we later get to know facts that conflict with these assumptions, we need to retract the conclusions made before. Because of the monotonicity of logical deduction, such a reasoning behavior cannot be modeled within classical logical systems. A less strict type of inference is necessary, that is known as non-monotonic reasoning. We define a set of relations that capture this kind of inference:

Definition 1 ((Non-monotonic) Consequence Relation)
A consequence relation is a binary relation $\mathrel{|\!\sim}$ on a set of formulas that contains a pair $(A, B) \in \mathcal{L}_\Sigma \times \mathcal{L}_\Sigma$ if B can be inferred from A.
A consequence relation is said to be non-monotonic, if it does not fulfill the property of monotonicity, defined as

$$\textit{if } A \mathrel{|\!\sim} B \quad \textit{then } A \wedge C \mathrel{|\!\sim} B,$$

for every $C \in \mathcal{L}_\Sigma$ with $A, B \in \mathcal{L}_\Sigma$.

The exact definition of the conditions under which a formula B is inferred from A will be given separately for each knowledge representation system analyzed in this thesis, as it depends on the specific semantics. While this negative characterization leaves room for many possible non-monotonic consequence relations, later in this thesis positive characterizations of a desired consequence relation will be discussed. The aim is to find a system for knowledge representation that provides a consequence relation with a reasoning behavior similar to that of human reasoning.

2.2 Intuition and syntax of conditionals

Over the course of the last decades, numerous formalisms for non-monotonic reasoning have been suggested and studied. An introduction to the oldest and most prominent systems can be found in text books such as [BHS93], [Bre91] and [Łuk90], while newer approaches are only documented in journal papers and conference proceedings of the field. The core concept of almost any of these formalisms is that we can define *default rules* which do not need to hold in every circumstances but can have exceptions.

As Bourne points out in [Bou99], there are two underlying approaches behind the known formalisms. In the *extensional approach*, non-monotonic reasoning is formalized by default rules that can be applied to a knowledge base under specific circumstances only, creating so-called extensions. Examples of this approach are default logic and circumscription. A disadvantage is that this reasoning process can lead to several extensions with no preference for one of them. In order to obtain desired conclusions, the reasoning has to be guided manually by blocking the application of some defaults explicitly. In the *conditional approach*, defaults are interpreted as constraints on a consequence relation, ensuring that at least the knowledge base itself is part of the relation. This avoids the complications of the first approach when dealing with exceptional circumstances. However, the challenge here is to define what other inferences, in addition to those given by the constraints, should be part of the consequence relation. In this thesis, formalisms following the latter approach are discussed.

A *conditional* is a formula $(B|A)$ with the intuitive meaning that, if A holds, we know that usually B holds as well, but that exceptions are possible. Or in other words, as Kern-Isberner puts it, conditionals are "plausible yet defeasible conclusions" and "represent generic knowledge, acquired inductively from experience or learned from books" [KI01b, p. 604]. A is called *antecedence* and B is the *consequence*. As an example, consider

$$(\, dangerous \,|\, bear \,)$$

with the intended meaning that bears are usually dangerous. If we know about an animal that it is a bear, we can jump to the conclusion that it is dangerous using this conditional. The same would be possible with the formula $bear \Rightarrow dangerous$ in classical logic.

However, if we discover that our animal is in fact a teddy bear, we might want to retract the previous conclusion, a type of reasoning that is not allowed in classical logic. The practice of using rules that hold in general but can have some exceptions is known as *default reasoning*, and default reasoning is one of the main use cases of non-monotonic consequence relations (cf. [Bre91, p. 5–7]). In the next chapter, different semantics for conditionals are presented that allow this kind of reasoning.

Within this thesis, we use the following notation for logical formalisms: Σ_{PL} denotes a propositional signature consisting of a finite set of propositional variables, denoted by upper-case letters. The set of formulas $\mathcal{L}_{\Sigma_{PL}}$ is built on it in the usual way, using conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), material implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$). Conjunctions will be abbreviated by omitting $\wedge$ and negations by overlining. $\top$ denotes a formula that always holds, while $\bot$ can never be true. This propositional language is extended to the language of *propositional conditionals* ($\mathcal{L}_{\Sigma_{PL}} \mid \mathcal{L}_{\Sigma_{PL}}$) by the operator "|", creating conditionals $(B|A)$ with $A, B \in \mathcal{L}_{\Sigma_{PL}}$. Conditionals cannot be nested. A finite set $\mathcal{KB}_{\Sigma_{PL}}$ of conditionals is called a *propositional knowledge base*.

In the first-order setting, let Σ_{FOL} be a restricted first-order signature consisting of a finite set of predicate symbols $\mathcal{P}$, a finite set of constant symbols D but no function symbols with higher arity.[1] An atom is a predicate with a list of constants or variables corresponding to its arity. The formulas of the first-order language $\mathcal{L}_{\Sigma_{FOL}}$ are built from atoms with the same connectives as in the propositional case, but with existential ($\exists$) and universal ($\forall$) quantification being available as well. In a closed formula, all variables are bound by quantifiers, otherwise, the formula is open. For an open formula $A(x)$, x is the vector of all free variables in A. $A(c)$ denotes the instantiation of those variables with the constants in vector c, requiring c and x to have the same length.

Similar to the propositional case, a language of *first-order conditionals* is created by introducing the conditional operator. In addition to conditionals $(B|A)$ with $A, B \in \mathcal{L}_{\Sigma_{FOL}}$, the language ($\mathcal{L}_{\Sigma_{FOL}} \mid \mathcal{L}_{\Sigma_{FOL}}$) also includes (outer) universally or existentially quantified conditionals $\forall x(B|A)$ and $\exists x(B|A)$. Again, conditionals cannot be nested. A *first-order knowledge base* $\mathcal{KB}_{\Sigma_{FOL}} = \langle \mathcal{F}, \mathcal{R} \rangle$ consists of a finite set $\mathcal{R}$ of first-order conditionals and a finite set $\mathcal{F}$ of closed first-order formulas, called facts.

Examples for theses kinds of formulas and conditionals will be given in the next sections, as semantics for them are introduced. If it is clear from the context, the reference to either the propositional or the first-order case will be omitted in the notation.

[1] Such a subset of first-order logic is also known as *relational logic*, as only relations (predicates) are used. Throughout the thesis, following [KIB15], the general term first-order is used though.

2.3 Semantics of propositional conditionals

Having introduced conditionals, their syntax and the underlying logical languages before, semantics are given to them now. In this section, we focus on the propositional case and outline the features of the approach central to this thesis, which is system Z by Pearl. The most important definitions and theorems are reproduced from the literature in order to compare them to their first-order counterparts later.

In contrast to classical logical languages, in which a formula either holds or not, conditionals have a three-valued nature. If, for a conditional $(B|A)$, A as well as B holds, we say the conditional is *verified*. If A holds but B does not, the conditional is *falsified*. If however, the antecedence A itself does not hold, the conditional is *not applicable* and neither verified nor falsified. In order to check whether the antecedence or the consequence holds, interpretations and satisfaction of the underlying language are used. In our propositional logic $\mathcal{L}_{\Sigma_{PL}}$, the interpretations $Int(\Sigma_{PL})$ are assignments of truth values to all propositional variables of the signature, and $\models_{\Sigma_{PL}}$ the satisfaction relation based on these interpretations. By Ω, we denote the set of possible worlds that arises from possible truth-assignments. To each of these worlds, a ranking function assigns an integer value.

Definition 2 (Ranking Function)
A ranking function κ is a function $\kappa : \Omega \to \mathbb{N} \cup \{\infty\}$ with $\kappa^{-1}(0) \neq \emptyset$.

Spohn introduced the concept of ranking functions, also called ordinal conditional functions, as "a grading of disbelief in possible worlds" [Spo88, p. 155]. A world with a lower rank is assumed to be more *natural* or *plausible* than a world with a higher rank, and the definition requires that there is at least one world which is ranked with 0 and therefore a most plausible one. Based on a ranking of possible worlds, ranks can also be assigned to formulas and conditionals. Formulas are ranked by finding the minimal of all possible worlds in which they are valid. A conditional is ranked with the difference between the ranks of its verification formula and its antecedence.

Definition 3 (Ranking of Formulas and Conditionals)
A formula $A \in \mathcal{L}_{\Sigma_{PL}}$ and a conditional $(B|A) \in (\mathcal{L}_{\Sigma_{PL}} \mid \mathcal{L}_{\Sigma_{PL}})$ are ranked

$$\kappa(A) = \begin{cases} \infty & \text{if there is no } \omega \in \Omega \text{ with } \omega \models A \\ \min\{\, \kappa(\omega) \mid \omega \in \Omega,\, \omega \models A \,\} & \text{otherwise} \end{cases} \tag{2.1}$$

$$\kappa(B|A) = \begin{cases} \infty & \text{if } \kappa(AB) = \infty \\ \kappa(AB) - \kappa(A) & \text{otherwise} \end{cases} \tag{2.2}$$

by a ranking function κ on possible worlds Ω.

B	T	D	$\kappa(\omega)$	B	T	D	$\kappa(\omega)$
0	0	0	0	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1
0	1	1	1	1	1	1	0

Figure 2.1: Ranking function κ of Example 1

Ranking functions provide a semantic interpretation for conditionals similar to the satisfaction relation defined in classical logical languages. In fact, we use the same symbol $\models$ for all of these relations. In the case of a ranking function, this relation is called *acceptance relation* and is defined for formulas, conditionals and sets of conditionals.

Definition 4 (Acceptance Relation)
The acceptance relation $\models$ of a ranking function κ for propositional formulas A, conditionals $(B|A)$ and knowledge bases $\mathcal{KB}$ is defined as

$$\kappa \models A \text{ iff for all } \omega \in \Omega \text{ with } \kappa(\omega) = 0 : \omega \models A \tag{2.3}$$

$$\kappa \models (B|A) \text{ iff } \kappa(AB) < \kappa(A\overline{B}) \tag{2.4}$$

$$\kappa \models \mathcal{KB} \text{ iff for all } (B|A) \in \mathcal{KB} : \kappa \models (B|A) \tag{2.5}$$

*If a knowledge base $\mathcal{KB}$ is accepted by a ranking function κ, we call κ a **model** of $\mathcal{KB}$. A knowledge base is **inconsistent** if it has no model.*

Basically, a conditional is accepted if its verification is more plausible than its falsification. The acceptance therefore depends on the rank that the ranking function assigns to each world. A brief example illustrates this idea:

Example 1 We observe an animal. Let $\mathcal{KB}_1 = \{(D|B), (B|T)\}$ be our knowledge base. The first conditional says that bears are usually dangerous and the second conditional says that teddy bears are usually bears. In Figure 2.1, a ranking function κ is shown that assigns a rank to each of the 8 possible worlds in this example. We can easily see that $\kappa(T\overline{B}) = 1$ since both worlds that satisfy $T\overline{B}$ have the rank 1. The worlds $bt\overline{d}$ and btd are ranked 1 and 0, so $\kappa(TB)$ is the minimum of $\{0, 1\}$, which is 0. The conditional $(B|T)$ is therefore accepted, the necessary condition $0 < 1$ holds. Because $(D|B)$ is also accepted, the knowledge base as a whole is accepted and κ is a model of $\mathcal{KB}_1$. ∎

Using this concept of a model of a knowledge base, we can now look at consequence relations that are based on ranking functions. The idea is that from a formula another formula can be inferred in the context of a given knowledge base, if the corresponding conditional is accepted by the ranking function. The question is which of the possible

ranking functions should be used. In [Ada75] Adams proposed a first definition that takes all ranking functions into account that are a model of the knowledge base:

Definition 5 (P-Entailment)
Let $\mathcal{KB}$ be a knowledge base. A formula B can be p-entailed from A, denoted by $A \mid\sim_p B$, if and only if it is accepted by all models of the knowledge base:

$$A \mid\sim_p B \text{ iff } \kappa \models (B|A) \text{ for all } \kappa \text{ with } \kappa \models \mathcal{KB}$$

P-entailment was also named ϵ- and r-entailment by other authors (cf. [Pea90, p. 124]). In order to test whether a formula B is p-entailed from A, we do not need to create all models of the knowledge base. An equivalent definition of p-entailment is that

$$B \text{ is p-entailed from } A \text{ iff } \mathcal{KB} \cup \{(\overline{B}|A)\} \text{ is inconsistent.}$$

Algorithm ConsistencyCheck can be used to test whether a knowledge base is consistent and thereby provides a way of implementing p-entailment. This important result for propositional knowledge bases is captured in the first theorem:

Theorem 1 (Consistency Check)
Algorithm ConsistencyCheck *creates a partition for a knowledge base $\mathcal{KB}$ if and only if $\mathcal{KB}$ is consistent. (cf. [Pea90],[GP96])*

Theorem 1 assures that the algorithm always finds a partition $\mathcal{R}_0 \cup ... \cup \mathcal{R}_m$ of the conditionals $\mathcal{KB}$ if the knowledge base is consistent and that the algorithm fails if the knowledge base is in fact inconsistent. In the created partition, each conditional is placed in a subset $\mathcal{R}_i$ such that it is tolerated by the conditionals in $\cup_{i \geq j} \mathcal{R}_j$. The notion of tolerance in this context is defined in the following way:

Definition 6 (Tolerance)
Let $\mathcal{KB}' \subseteq \mathcal{KB}$ be a subset of a knowledge base. A conditional $r \in \mathcal{KB}'$ is tolerated by the set $\mathcal{KB}'$ if a world exists in which r is verified and no other conditional of $\mathcal{KB}'$ is falsified.

Moreover, this partition can be used to obtain a specific ranking function, which is the idea of *system Z* developed by Pearl. [Pea90] is the central paper for the summary given here and also mentions the concepts we discussed so far in this section, though some of them appeared in other papers before. The following definition shows how a ranking function can be obtained from the partition of a knowledge base:

Definition 7 (System Z)
Let $\mathcal{KB}$ be a knowledge base with conditionals $r_i, 1 \leq i \leq n$, that are partitioned by algorithm ConsistencyCheck *as $\mathcal{R}_0 \cup ... \cup \mathcal{R}_m$. A function $Z : \mathcal{KB} \to \mathbb{N}$ assigns a z-rank*

Algorithm 1 ConsistencyCheck

Input: Knowledge base $\mathcal{KB}$

Output: Partition $\mathcal{KB} = \mathcal{R}_0 \cup ... \cup \mathcal{R}_m$ iff $\mathcal{KB}$ is consistent

1: $i \leftarrow 0, \mathcal{KB}_{left} \leftarrow \mathcal{KB}$
2: **while** $\mathcal{KB}_{left} \neq \emptyset$ **do**
3: find every $r \in \mathcal{KB}_{left}$ tolerated by $\mathcal{KB}_{left}$, add r to $\mathcal{R}_i$
4: **if** $\mathcal{R}_i = \emptyset$ **then**
5: **return** $\mathcal{KB}$ is inconsistent
6: **else**
7: $\mathcal{KB}_{left} \leftarrow \mathcal{KB}_{left} \setminus \mathcal{R}_i, i \leftarrow i + 1$
8: **return** $\mathcal{KB} = \mathcal{R}_0 \cup ... \cup \mathcal{R}_m$

to every conditional $r_i = (B_i|A_i)$ *based on the subset it is assigned to:* $Z(r_i) = j$ *iff* $r \in \mathcal{R}_j$. *Then the ranking function* κ_z *is given as*

$$\kappa_z(\omega) = \begin{cases} 0 & \text{if no } r_i \in \mathcal{KB} \text{ is falsified in } \omega \\ \max_{1 \leq i \leq n}\{Z(r_i) \mid \omega \models A_i\overline{B_i}\} + 1 & \text{otherwise.} \end{cases}$$

Each world is assigned a rank that depends on the set of conditionals which are falsified in that world. If the set is empty, the world receives the ranking 0. If not, the rank will be 1 plus the highest rank of the conditionals in the set. The ranking computed in this way has some interesting properties: It is a model of the knowledge base, and among all models, it is the minimal one.

Theorem 2 (Minimality of κ_z)
Let $\mathcal{KB}$ *be a knowledge base and* κ_z *be the ranking function computed according to Definition 7.* κ_z *is the unique minimal ranking function accepting the knowledge base.*

Minimal in this context means that if only one world had a lower ranking, the ranking function would no longer be a model of the knowledge base. Only one unique model satisfying the property of minimality exists. A proof for this theorem is given in [Pea90]. Based on this special ranking function, a second consequence relation can be defined which is known as z-entailment.

Definition 8 (Z-Entailment)
Let $\mathcal{KB}$ *be a knowledge base. A formula* B *can be z-entailed from* A, *denoted by* $A \mid\sim_z B$, *if and only if it is accepted by the model* κ_z *of the knowledge base:*

$$A \mid\sim_z B \text{ iff } \kappa_z \models (B|A)$$

B	T	D	$\kappa(\omega)$	B	T	D	$\kappa(\omega)$
0	0	0	0	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	2	1	1	0	1
0	1	1	2	1	1	1	2

Figure 2.2: Ranking function κ_z of Example 2

The more restricted p-entailment defined before allows less conclusions, since they have to be admitted by every model of a knowledge base. That consequence relation is therefore known as the *conservative core*. With z-entailment, more formulas can be derived, and this models human reasoning more accurately. In Section 2.5, these differences are discussed in detail. The computation and reasoning behavior of system Z is illustrated in the next example, which is an extension of the last one:

Example 2 Let $\mathcal{KB}_2 = \mathcal{KB}_1 \cup \{(\overline{D}|T)\} = \{(D|B), (B|T), (\overline{D}|T)\}$ be our knowledge base. Again, the first conditional says that bears are usually dangerous and the second conditional says that teddy bears are usually bears. The third conditional states that teddy bears are usually not dangerous, making them an exceptional kind of bears.

Applying algorithm ConsistencyCheck to our knowledge base, we can only add $(D|B)$ to the first subset, since there is no world that verifies more conditionals and does not falsify any of them. $(B|T)$ and $(\overline{D}|T)$ can be put in the second subset. The resulting partition is

$$\mathcal{R}_0 = \{(D|B)\},\ \mathcal{R}_1 = \{(B|T), (\overline{D}|T)\}$$

and the knowledge base is therefore consistent. In Figure 2.2, the ranking function κ_z that can be computed based on this partition is shown.

With p-entailment, we can conclude $TB \mid\sim_p \overline{D}$, meaning that a bear which is a teddy bear is usually not dangerous. A consistency check on the knowledge base $\mathcal{KB}_2 \cup \{(D|TB)\}$ reveals that it is inconsistent. Z-entailment also allows us to conclude that an animal which is not a bear is usually not a teddy bear as well. $\kappa(\overline{B}\overline{T}) = 0$ is less than $\kappa(\overline{B}T) = 2$, hence $\kappa_z \models (\overline{T}|\overline{B})$ and therefore $\overline{B} \mid\sim_z \overline{T}$. Using p-entailment, this conclusion cannot be made, since $\mathcal{KB}_2 \cup \{(T|\overline{B})\}$ is still consistent. ∎

Although z-entailment seems to be somewhat superior to p-entailment, it still has some weaknesses. The main problem is that system Z does not sanction property inheritance for exceptional subclasses. While examples for problems like this will be discussed later in this chapter, we will now look at a few extensions that have been suggested to cope with the shortcomings of system Z.

A straightforward extension of system Z is *system* Z^+ described in [GP96], which makes use of variable-strength conditionals instead of the conditionals we used so far. By adding a degree of strength to each conditional, we can encode additional information in a knowledge base and influence the priority between different conditionals. The strengths are taken into account when evaluating the acceptance of a conditional, and corresponding notions of tolerance, consistency and a minimal ranking function follow. This extension is useful in scenarios such as the Nixon diamond, in which we have conflicting conditionals without specificity. Giving more strength to one of the conditionals resolves the conflict.

One reason for system Z's weaknesses is that the computation of κ_z uses only the rank of the highest violated conditional in each world. Several systems have been suggested that calculate a ranking function based on more information and that can therefore provide a consequence relation that better reflects the knowledge base. For example, *lexicographical closure* proposed by Lehmann in [Leh95] counts for a possible world how many conditionals of each subset of a partition are violated. A tuple like $(1, 3)$ is then assigned to a world, indicating that one conditional of $\mathcal{R}_0$ and three conditionals of $\mathcal{R}_1$ are violated in this world. An order defined on these tuples provides a ranking over wolds similar to a ranking function used in system Z.

Another approach that incorporates more information of a knowledge base is motivated by the probabilistic interpretation of conditionals. A variable-strength conditional $(B|A)[x]$ can be interpreted as a conditional probability as given in the equation $P(B|A) = x$. For a knowledge base, several possible probability distribution can be found, each of them satisfying all these equations. Among them, the one having the highest entropy is chosen, as it adds as little information as possible to the information given by the knowledge base. In [GMP93], this idea of *maximum entropy* is used in *system* Z^* to compute a suitable ranking function, but the approach is limited to a restricted set of knowledge bases. A generalization is done by Bourne in [Bou99]. A similar approach by Kern-Isberner based on the same idea are *c-representations* presented in [KI01b] and [KI01a]. These systems can successfully resolve some issues of system Z by using the sum of all violations of conditionals for the calculation of a ranking function instead of only the maximum.

2.4 Semantics of first-order conditionals

A weakness that all approaches presented in the previous section share are the limitations of the underlying propositional language. In most cases, as Delgrande points out, the expressiveness of propositional calculus is not sufficient:

> "In general, a propositional logic will be inadequate for modelling phenomena of interest, and so ultimately one will require the richer expressiveness of a first-order logic." [Del98, p. 106]

With a first-order language, we are able to make statements about specific individuals and classes of individuals at the same time. For example, while $B(bruno)$ expresses that an individual named Bruno is a bear, $B(x)$ refers to the class of bears as a whole. Moreover, we can use predicates of higher arities to describe complex relations between classes and individuals. Statements of these types are not possible in a purely propositional language. Thus, it seems to be worthwhile studying how semantics can be defined for first-order conditionals languages.

Several different semantics have been suggested and discussed in the literature. In [FHK96], Friedman et al. introduce semantics based on plausibility measures, which are an abstraction of different semantics known from the propositional case. Brafman presents an extension of classical first-order interpretations in [Bra97] that come with a ranking of all individuals of the domain in order to evaluate conditionals. The approach of Delgrande in [Del98] in contrast makes use of a modal semantic that provides a ranking of worlds, and in each of these worlds, some individuals are found to be exceptional. However, the details of these systems will not be discussed here. Instead, we focus on some general issues that arise when moving to a first-order setting.

One of the challenges of a first-order conditional language is the interaction between universal quantification and the conditional operator. We could express the general rule that *bears are usually dangerous* in our language $\mathcal{L}_{\Sigma_{FOL}}$ using outer quantifications:

$$\forall x(D(x)|B(x)) \tag{2.6}$$

But this formula actually says that *for each individual, it holds that if it is a bear, it is usually dangerous*. Due to the quantification, the conditional is evaluated for each instance. If we have a knowledge base $\mathcal{KB} = \{\, \forall x(D(x)|B(x)) \,,\, (\overline{D}(a)|B(a)) \,\}$, in which we added information about a particular bear a which is usually not dangerous, we can easily see that it must be inconsistent. Obviously, conditional (2.6) does not express the intended meaning. We rather want to make a statement about the class of bears that should hold in general, but that can at the same time be violated by some exceptional individuals, without rendering the whole knowledge base inconsistent

For this purpose, both Brafman and Delgrande introduce an additional, weaker type of universal quantification that we denote by $\forall_c$ following Delgrande. In contrast to $\forall$, which ranges over the whole domain D of our signature, the quantifier $\forall_c$ only ranges over *normal* or *unexceptional* individuals of the domain. Using this type of quantification, we can express the intended default rule:

$$\forall_c x(D(x)|B(x)) \tag{2.7}$$

If we replace (2.6) by (2.7) in our knowledge base, it is consistent as long as we consider a to be an exceptional individual. The distinction between these types of quantifications is also relevant for the principle of *universal instantiation*. In case of a full universal quantification, we can substitute any individual of the domain for x and obtain, for

example, $(D(a)|B(a))$ for any $a \in D$. When dealing with the weaker quantification $\forall_c$ though, only unexceptional individuals can be substituted.

A second challenge in the first-order setting is that there are two different ways of interpreting a conditional. The one known from the propositional case is that $(B(x)|A(x))$ is true if in the most plausible worlds in which $A(x)$ holds, $B(x)$ holds as well. This makes the statement a *subjective conditional* whose semantical interpretation is a preference relation over possible worlds. The other reading says that the conditional holds if for most of the individuals, for which $A(x)$ holds, also $B(x)$ holds. A ranking over individuals is evaluated here, and statements like this are known as *statistical conditionals*. Friedman et al. and Brafman point out that a precise distinction between these interpretations has to be made when defining semantics for first-order conditionals (cf. [FHK96], [Bra97]).

A popular example which illustrates that subjective conditionals alone are not sufficient is the *lottery paradox*. In a typical lottery, every individual participating usually does not win the lottery. However, there usually is one who wins.

$$\{\, \forall x(\overline{W}(x)|\top)\,,\, (\exists x W(x)|\top)\,\} \tag{2.8}$$

In Brafman's approach, which deals with statistical conditionals only, this knowledge base can be consistently represented. In many systems with subjective semantics though, the formula $(\forall x\overline{W}(x)|\top)$ can be derived, which clearly conflicts with the second conditional. Even if the quantifier $\forall$ is replaced by $\forall_c$ in the first conditional, the knowledge base remains inconsistent. This happens for example in Delgrande's logic, forcing him to admit that statistical conditionals are necessary to model this situation:

> "What this shows is not that the present approach is inadequate, but rather that the standard lottery paradox is not plausibly represented using notions of 'normality' that are not probabilistically based. In stating that 'normally one does not win the lottery', the sense of 'normal' is that of statistical likelihood." [Del98, p. 121]

2.5 Benchmark problems and desirable properties

After the last sections presented several approaches for non-monotonic default reasoning with conditionals, and as there are many other approaches available in the literature, an obvious question is how they can be compared and which one should be preferred. A popular technique in this field is to evaluate how certain *benchmark problems* are handled by the systems. Another, more formal, method of evaluation is the *classification of consequence relations* by formal properties. Both techniques will be shown here in order to structure the content provided in the last sections but also to lay the ground for the analysis that is done in the upcoming chapters.

Benchmark problems are simple knowledge bases that can be used to test if a desired reasoning behavior is provided by a system of default reasoning. Although the use of benchmark problems can be found in almost any paper of this field, there is no agreed list of them. In [Lif89], Lifschlitz made an attempt to compile such a list. Within this thesis, we use the list Bourne presented in chapter 5 of [Bou99]. It contains problems for desired reasoning behavior that are based on "a broad consensus" and that can be found in most of the papers cited so far.

One of the most basic behaviors that we would expect from a system of default reasoning is that of *property inheritance*. We use conditionals to express default rules such as that bears are usually dangerous $(D|B)$, which hold in general but can have exceptions. Let's assume we have a second rule saying that grizzlies are usually bears $(B|G)$, making grizzlies a subclass of the class of bears. In a system that sanctions property inheritance, we should be able to conclude that grizzlies are usually dangerous $(D|G)$, unless there is specific knowledge about grizzlies being different in this aspect. While the term property inheritance already provides a semantic interpretation of this reasoning behavior, more neutral terms that are also commonly used are transitivity and rule chaining. A second benchmark problem tests whether the desired reasoning behavior is also obtained in the presence of *irrelevant facts*. Extending the example above, we could ask whether a red bear is usually dangerous $(D|RB)$, and if we do not have any specific information about the property of being red R, the conclusion should hold.

More interesting problems evaluate the reasoning behavior in case of conflicting conditionals. Assume we know that bears are usually dangerous $(D|B)$, teddy bears are usually bears $(B|T)$ but they are usually not dangerous $(\overline{D}|T)$. In this case, the property of being dangerous should not be inherited from bears to the subclass of teddy bears because there is explicit information about teddy bears not being dangerous. The conflict between the conditionals $(D|B)$ and $(\overline{D}|T)$, that both apply for a teddy bear T, should be resolved in favor of the more specific one $(\overline{D}|T)$, and this behavior is called *specificity*. However, the fact that teddy bears are special regarding their dangerousness should not prevent other properties of bears being inherited to teddy bears. If we have another conditional $(P|B)$, saying that bears usually have paws, we also expect teddy bears to have paws unless we have information that says the opposite. In other words, the conditional $(\overline{D}|T)$ should not make the teddy bear as a whole exceptional but only with regard to that single property. Every other property should be inherited as usual, and this desired behavior is known as *exceptional inheritance*, while its absence is called the *drowning problem*.

Another popular benchmark problem is the aforementioned Nixon diamond. It tests how conflicting conditionals are handled if none of them is more specific than the other. The original example here is that we know that quakers usually are pacifists and republicans usually are not. In case of Nixon, who is both a quaker and a republican, both conditionals are applicable but conflict with each other. Since there is no preference for one of the conditionals, a proper system for default reasoning should neither conclude that Nixon is a pacifist nor that he is the opposite.

For the default reasoning systems introduced in Section 2.3, the handling of the benchmark problems is the following: p-entailment sanctions property inheritance and specificity, but cannot handle irrelevant facts and exceptional inheritance. z-entailment resolves the issue with irrelevance, but still fails to support exceptional inheritance (cf. [Pea90]). System Z^+ is similar to System Z, but it can resolve problems such as the Nixon diamond in favor of one of the conflicting conditionals if the one is given a greater strength (cf. [GP96]). Only lexicographical entailment as well as the approaches based on maximum entropy can additionally cope with the drowning problem (cf. [Bou99]) and therefore handle all benchmark problems as desired. In Chapter 5 we evaluate how the new system analyzed in this thesis handles the benchmark problems.

The second, more formal approach to evaluate the reasoning behavior of a default reasoning system is to analyze whether the obtained consequence relation satisfies desired inference patterns. In Definition 1 we introduced a non-monotonic consequence relation as not being monotonic. Now, we take a closer look on the positive properties that define non-monotonic consequence relations of interest. An influential paper following this approach is [KLM90], in which Kraus et al. define different classes of consequence relations, among them the class of preferential consequence relations which is relevant in this context. Technically, they provide a set of inference rules in order to define the class of all consequence relations satisfying these rules.

Definition 9 (Preferential Consequence Relation)
A consequence relation satisfying the following rules is a preferential consequence relation:

(1) *Reflexivity* $A \mid\sim A$

(2) *Left Logical Equivalence* $$\frac{\models A \Leftrightarrow B\,,\ A \mid\sim C}{B \mid\sim C}$$

(3) *Right Weakening* $$\frac{\models A \Rightarrow B\,,\ C \mid\sim A}{C \mid\sim B}$$

(4) *Cautious Monotonicity* $$\frac{A \mid\sim B\,,\ A \mid\sim C}{A \wedge B \mid\sim C}$$

(5) *And* $$\frac{A \mid\sim B\,,\ A \mid\sim C}{A \mid\sim B \wedge C}$$

(6) *Or* $$\frac{A \mid\sim C\,,\ B \mid\sim C}{A \vee B \mid\sim C}$$

Kraus et al. argue that every consequence relation used to model default reasoning should satisfy these rules and there is a broad consensus for this in the research community. p-entailment as well as z-entailment are both preferential consequence relations. In fact, as Kern-Isberner and Eichhorn point out in [KIE14], every consequence relation induced by a ranking function is a preferential consequence relation.

Although the rules in Definition 9 impose reasonable restrictions, the class of preferential consequence relations still contains some relations that do not represent a desired reasoning behavior. Especially, in some of them irrelevant facts are not handled as described above. Kraus et al. therefore suggest to add an additional rule to define the more restricted class of rational consequence relations, that, among other things, ensure that irrelevance is handled as desired.

Definition 10 (Rational Consequence Relation)
A preferential consequence relation satisfying the rule of Rational Monotonicity

$$\frac{A \mid\sim C\,,\; A \not\mid\sim \overline{B}}{A \wedge B \mid\sim C} \tag{2.9}$$

is called a rational consequence relation.

While the rules of Definition 9 provide a precise way of how to extend a given knowledge base to a preferential consequence relation, the rule of Rational Monotonicity is more difficult. Since the rule requires that certain inferences *cannot* be made, we can, if a consequence relation contains $A \mid\sim C$, add either $A \mid\sim B$ or $AB \mid\sim C$ to it, but not both. In general, a knowledge base therefore can have several extensions that are rational consequence relations. Among them, Lehman and Magidor identify a preferable one, which they call the *rational closure* (cf. [LM92]). Every ranking function that is a model of a knowledge base induces a rational consequence relation, and the consequence relation obtained from the ranking function of system Z is exactly the rational closure. Hence, system Z and rational closure are equivalent approaches (cf. [Pea90]).

Based on this fundamental analysis of properties and classes of consequence relations done by Kraus et al. and others, consequence relations obtained from specific systems can be evaluated more properly. Instead of testing benchmark problems, it can be formally proven whether the inference behavior follows desired rules or not. However, even the notion of rational consequence relations does not fully capture the reasoning we want to model, as such relations do not necessarily sanction exceptional inheritance.

3 First-Order System Z

In this chapter, the system for default reasoning that is the object of investigation in this thesis is introduced. The new approach, called system Z^{FOL} in this thesis, has been developed by Kern-Isberner and Beierle and is an extension of system Z to a first-order language. First, the definition of a ranking function given in the previous chapter is extended to deal with closed and open first-order formulas and conditionals. Then, the suggested construction procedure for the ranking function is presented and illustrated by an example. Finally, open questions regarding the approach that are addressed in the following chapters are stated.

3.1 First-order ranking functions

As in the propositional case, a ranking function assigns a degree of plausibility to each possible world. In our restricted first-order language $\mathcal{L}_{\Sigma_{FOL}}$, an interpretation from $Int(\Sigma_{FOL})$ consists of a domain of individuals and actual interpretations of the used predicates and constants on this domain. We make use of Herbrand interpretations and let the constants be interpreted by itself. That means, the set of constants D, which is also the Herbrand universe as we have no other function symbols, is the domain in any interpretation. Therefore, both the set of constants and the domain will be denoted by D, and, in later examples, if we define a domain, we thereby also introduce the constants available in the signature. All possible instantiations of predicates with corresponding constants built the set of ground atoms, known as the Herbrand base $\mathcal{H}$. Each subset of $\mathcal{H}$ is a possible world in which exactly the contained atomic formulas hold. We denote these worlds by giving a formula that combines all ground atoms in a conjunction, in which those that are not valid in the world occur as a negated atom. Again, the set of possible worlds will be denoted by Ω. The satisfaction relation $\models_{\Sigma_{FOL}}$ indicates that a first-order formula is valid in a world, for example, $\omega \models A$, and on this level, we usually work with closed formulas only in this first-order approach. Hence, we can directly see whether the relevant ground atoms are valid in the world or not.

Ranking functions in the sense of Definition 2 are now defined for first-order worlds. The definition follows [KIT12], which is a preceding paper of [KIB15] by Kern-Isberner and Thimm that shows how ranking functions can be defined in the first-order case and be interpreted by c-representations. Closed formulas are ranked as propositional formulas by finding the minimum rank of the worlds in which they are valid. In a similar way, closed

first-order conditionals are handled. For open formulas and conditionals though, every possible instantiation is evaluated and the minimum rank is taken.

Definition 11 (Ranking of First-Order Formulas and Conditionals)
Let $A \in \mathcal{L}_{\Sigma_{FOL}}$ be a first-order formula and $(B|A) \in (\mathcal{L}_{\Sigma_{FOL}} \mid \mathcal{L}_{\Sigma_{FOL}})$ be a first-order conditional. Closed formulas and conditionals are ranked according to Definition 3. Open formulas and conditionals are ranked

$$\kappa(A(x)) = \begin{cases} \infty \text{ if } \kappa(A(a)) = \infty \text{ for all } a \in \mathcal{H}^{A(x)} \\ \min\{\, \kappa(A(a)) \mid a \in \mathcal{H}^{A(x)} \,\} \text{ otherwise} \end{cases} \tag{3.1}$$

$$\kappa(B(x)|A(x)) = \begin{cases} \infty \text{ if } \kappa(B(a)|A(a)) = \infty \text{ for all } a \in \mathcal{H}^{(B(x)|A(x))} \\ \min\{\, \kappa(B(a)|A(a)) \mid a \in \mathcal{H}^{(B(x)|A(x))} \,\} \text{ otherwise} \end{cases} \tag{3.2}$$

by a ranking function κ on possible worlds Ω.

By $\mathcal{H}^{A(x)}$, we denote the set of all constant vectors that can be used to instantiate the formula $A(x)$, with x being the vector of free variables. x and a must be vectors of the same length. $\mathcal{H}^{(B(x)|A(x))}$ is the corresponding set for a conditional.

Based on the ranks, the definition of the acceptance relation is extended to first-order formulas and conditionals as well. We differentiate several cases here: Closed formulas and conditionals are accepted as in the propositional case. The acceptance of outer quantified conditionals is evaluated against the set of possible instantiations in the usual way. So a conditional $\exists x(B(x)|A(x))$ is accepted by κ if at least one possible instantiation is accepted. The most interesting case however is the acceptance of open conditionals.

Definition 12 (Acceptance of First-Order Formulas and Conditionals)
The acceptance relation $\models$ of a ranking function κ for first-order formulas A and first-order conditionals $(B|A)$ is defined as follows:

- *for closed formulas and conditionals*
  - *according to Definition 4*
- *for conditionals with outer quantifications*
  - $\kappa \models \forall x(B(x)|A(x))$ *iff* $\kappa \models (B(a)|A(a))$ *for all* $a \in \mathcal{H}^{(B(x)|A(x))}$
  - $\kappa \models \exists x(B(x)|A(x))$ *iff there is* $a \in \mathcal{H}^{(B(x)|A(x))}$ *such that* $\kappa \models (B(a)|A(a))$
- *for open conditionals*
  - $\kappa \models (B(x)|A(x))$ *iff* $Rep((B(x)|A(x))) \neq \emptyset$ *and*

(Acc-1) $\kappa(A(x)B(x)) < \kappa(A(x)\overline{B}(x))$ *holds or*

(Acc-2) $\kappa(A(x)B(x)) = \kappa(A(x)\overline{B}(x))$ *and for all* $a \in Rep((B(x)|A(x)))$ *and for all* $b \in Rep((\overline{B}(x)|A(x)))$ *it holds that* $\kappa(A(a)\overline{B}(a)) < \kappa(A(b)B(b))$.

The intended meaning of an open conditional is that of a default rule. If we have the predicates B (being a bear) and D (being dangerous), the conditional $(D(x)|B(x))$ express that members of the class of bears are usually dangerous. In order to evaluate this, as we already discussed in Section 2.4, we do not want to check that the conditional holds for every possible bear. Instead, only normal bears, that are typical of the class of bears, should be taken into account. This idea was called a weak quantification and denoted as $\forall_c x(D(x)|B(x))$ in Section 2.4. In the semantics presented here, every open conditional is implicitly interpreted as being a weak quantification and we therefore omit $\forall_c$.

The set of individuals that is relevant to evaluate the acceptance of an open conditional is defined in the following way:

Definition 13 ((Weak) Representative)
Let $r = (B(x)|A(x)) \in (\mathcal{L}_{\Sigma_{FOL}} \mid \mathcal{L}_{\Sigma_{FOL}})$ *be an open first-order conditional. The set of weak representatives* $WRep(r)$ *contains all* $a \in \mathcal{H}^{(B(x)|A(x))}$ *satisfying the following conditions:*

$$\kappa(A(a)B(a)) = \kappa(A(x)B(x)) \tag{3.3}$$

$$\kappa(A(a)B(a)) < \kappa(A(a)\overline{B}(a)) \tag{3.4}$$

(Strong) representatives $Rep(r)$ *are all* $a \in WRep(r)$ *that satisfy*

$$\kappa(A(a)\overline{B}(a)) = \min\{\, \kappa(A(b)\overline{B}(b)) \mid b \in WRep(r) \,\}. \tag{3.5}$$

Weak representatives of an open conditional are individuals for which the conditional is accepted, as it is required by condition (3.4). Among them, according to condition (3.3), only the ones which are most plausible, in the sense of having the lowest rank, are chosen. Given this set of weak representatives, every individual in it that is violated in the lowest ranked world is a representative. The rationale behind condition (3.5) is that we expect the violation of the conditional for a typical individual to be more natural than for a more specific individual, as the latter would also be violated for a very specific reason. This characterizes representatives of a conditional as being individuals which *accept* the conditional and which are both *most plausible* and *least exceptional*.

In our bear scenario, consider a bear a, a normal teddy bear b and a teddy bear c which has a broken glass eye. We want to find an individual which is a representative for the conditional $(D(x)|B(x))$ saying that bears are usually dangerous. The normal teddy bear b should not be a representative since it is usually not dangerous. Condition (3.4) ensures this. The bear a and the broken teddy bear c both satisfy (3.4), but among them, only

the more general individual should be a representative. Both (3.3) and (3.5) express this idea of being most general in a positive and negative way. In our example, the broken teddy bear c could violate the conditional by not being dangerous because the broken glass eye does not have any sharp edges. This is obviously a very specific reason, whereas the bear a could violate the conditional in a more general way. Condition (3.5) would therefore qualify a to be the representative in this case.

Based on the notion of representatives, the acceptance of an open conditional is defined in Definition 12. The first condition says that there must be at least one representative, otherwise, the conditional cannot be accepted. Then, it is checked whether the verification of the conditional is more plausible than its falsification in *(Acc-1)*. If that is the case, it is accepted, if not, *(Acc-2)* is checked. In this case, the conditional is accepted if it is falsified – being instantiated with its representatives – more plausibly than it is verified, being instantiated with the representatives of the negated conditional. This last condition again follows the idea of least exceptional violation.

The remaining type of formula for which we did not yet define acceptance are open formulas $A(x)$. They can be expressed as conditionals with tautological antecedence $(A(x)|\top)$ and then be handled as open conditionals. In Section 2.2, we already introduced the notion of a first-order knowledge base. In contrast to the propositional case, we allow it to contain conditional as well as certain knowledge, being separated into two sets. The next definition captures this idea:

Definition 14 (First-Order Knowledge Base)
A first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ consists of a finite set $\mathcal{R}$ of first-order conditionals and a finite set $\mathcal{F}$ of closed first-order formulas, called facts.
In a restricted first-order knowledge base, only unary predicates may occur, and a formula or conditional may not contain more than one variable or constant. Further, no outer quantifications are allowed in $\mathcal{R}$.

The second, more restricted kind of a knowledge base will be relevant in the upcoming sections. For the definition of first-order ranking functions, however, the more general version applies. The acceptance of such a knowledge base is not only based on the conditionals but also requires the worlds falsifying the facts to have the rank ∞.

Definition 15 (Acceptance of a First-Order Knowledge Base)
The acceptance relation $\models$ of a ranking function κ accepts a first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ iff every conditional in $\mathcal{R}$ is accepted and $\kappa(\omega) = \infty$ for each ω with $\omega \not\models \mathcal{F}$.

The terms *model* and *inconsistent* are used as in the propositional case. Now, we have a full definition of semantics for first-order conditionals. Before we move on to the construction of suitable ranking functions, a few remarks on the intended meaning of the formulas of our language conclude this section.

We can use the set $\mathcal{F}$ of a first-order knowledge base to express *certain* knowledge. Continuing with our running example, a formula $B(a) \in \mathcal{F}$ expresses that the individual a certainly is a bear. The quantified formula $\forall x B(x) \in \mathcal{F}$ says that every individual of the domain certainly is a bear. Open formulas such as $B(x)$ cannot be part of $\mathcal{F}$. In addition to this certain knowledge, *plausible* knowledge can be provided in $\mathcal{R}$. The conditional $(B(a)|\top)$ means that the individual a is usually a bear, and $(\forall x B(x)|\top)$ expresses that usually all individuals are bears. In contrast, the open conditional $(B(x)|\top)$ formalizes that normal individuals are usually bears, making use of the notion of representatives. And finally, a quantified conditional $\forall x(B(x)|\top)$ says that for every individual x it holds that x is usually a bear. With all these kinds of formulas and their different meanings at hand, we have a rich language available to accurately model knowledge bases.

3.2 Construction of ranking functions

In the propositional case, system Z offers a way to create a unique ranking function for a given knowledge base. Based on the extension of ranking functions to first-order knowledge bases discussed in the last section, a similar way of creating a ranking function is now introduced for the first-order case. It was published by Kern-Isberner and Beierle in [KIB15]. During the rest of this thesis, this approach will be tested and analyzed.

For the rest of this thesis, only knowledge bases of the restricted type defined in Definition 14 are relevant, as these limitations are crucial for the definition of system Z^{FOL}. Since we do not allow predicates with an arity other than one, and a formula or conditional may not contain more than one variable, formulas and conditionals are simplified. By writing $A(x)$, x is no longer a vector but a single variable and the formula A can be instantiated by a single constant. The absence of outer quantifications is actually not a restriction, because quantified conditionals can be replaced by their instantiations with constants from the finite domain.

In Definition 6, the notion of tolerance was introduced, and algorithm ConsistencyCheck has been presented that creates a partition of a propositional knowledge base in which each conditional is tolerated by the conditionals in the same or higher subsets. In the first-order setting, a more complex definition of tolerance is needed that also takes into account the domain of individuals.

Definition 16 (Tolerance-Pair)
Let $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a restricted first-order knowledge base and D be the set of constants respectively the domain. A pair of ordered partitions for the set of conditionals $\mathcal{R}$ and the domain D, denoted as $p = \langle (\mathcal{R}_0, D_0), \ldots, (\mathcal{R}_m, D_m) \rangle$, is a tolerance-pair of $\mathcal{KB}$ and D if and only if

$\forall i \in \{0, ..., m\} : \forall r \in \mathcal{R}_i : \exists\, a \in D_i$ *and* $\omega \in \Omega$ *with*

(1) $\omega \models \mathcal{F}$
(2) ω *verifies* $r(a)$
(3) ω *does not falsify* $r'(a')$ *for all* $r' \in \cup_{j \geq i} \mathcal{R}_j$ *and all* $a' \in D_i$.

By p_i*, we denote the tuple* $(\mathcal{R}_i, D_i)$ *of* p*, and by* $p_{\mathcal{R},i}$ *and* $p_{D,i}$ *its components.*

A tolerance-pair partitions both the set of conditionals as well as the domain of a first-order knowledge base. Conditions (2) and (3) are the first-order counterpart of the definition of tolerance. Note that while all conditionals in $\cup_{j \geq i} \mathcal{R}_j$ are relevant for (3) as in the propositional case, only the constants in D_i are taken into account. The additional condition (1) makes sure that the certain knowledge $\mathcal{F}$ is considered properly.

Based on such a tolerance-pair, a first-order version of system Z, named system Z^{FOL}, is defined in the following way by Kern-Isberner and Beierle:

Definition 17 (System $\mathbf{Z}^{FOL}$)
Let $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ *be a restricted first-order knowledge base,* D *be a domain and* p *be a tolerance-pair* $\langle (\mathcal{R}_0, D_0), \ldots, (\mathcal{R}_m, D_m) \rangle$ *for* $\mathcal{KB}$ *and* D*.*
A function λ_i *assigns to every world* $\omega \in \Omega$*,* $\omega \models \mathcal{F}$*, for* $i \in \{0, ..., m\}$*, an integer:*

$$\lambda_i(\omega) = \begin{cases} 0, \text{ if no } r(a) \in \mathcal{R} \text{ with } a \in p_{D,i} \text{ is falsified in } \omega \\ \max\limits_{a \in p_{D,i}} \max\limits_{r \in \mathcal{R}} \{\, j \mid r \in p_{\mathcal{R},j} \text{ and } r(a) \text{ is falsified in } \omega \} + 1, \text{ otherwise.} \end{cases}$$

Then the ranking function κ_z^{FOL} *is given as* $\kappa_z^{FOL}(\omega) = \infty$*, if* $\omega \not\models \mathcal{F}$*, and otherwise*

$$\kappa_z^{FOL}(\omega) = \sum_{i=0}^{m} (m+2)^i \lambda_i(\omega) - \kappa_0 \qquad \text{with} \qquad \kappa_0 = \min_{\omega \in \Omega} \sum_{i=0}^{m} (m+2)^i \lambda_i(\omega).$$

The function λ_i is defined similarly to κ_z in the propositional case, as it determines the rank of a world based on the highest-labeled subset that contains a falsified conditional. However, λ_i is defined per subset of the partition and instantiates conditionals with constants from that subset in order to check for verification and falsification. In κ_z^{FOL} the results of all subsets are combined in a sum. The term κ_0 normalizes the resulting function to ensure that it is a valid ranking function that assigns zero to at least one world.

In their paper, Kern-Isberner and Beierle proof that a ranking function obtained by Definition 17 for a knowledge base that satisfies the mentioned restrictions is always a model of that knowledge base. That means, κ_z^{FOL} accepts all conditionals in $\mathcal{R}$ and assigns ∞ to all worlds in which the certain knowledge $\mathcal{F}$ is not satisfied.

Theorem 3 (Kern-Isberner and Beierle)
Let $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a restricted first-order knowledge base with $\mathcal{F}$ being consistent. Let p be a tolerance-pair of $\mathcal{KB}$ and the domain D.
Then the ranking function κ_z^{FOL} induced by p is a model of $\mathcal{KB}$.

Based on the ranking function, we can define a consequence relation that represents a set of formulas which can be derived from a first-order knowledge base. The definition is completely analogous to the propositional case.

Definition 18 ($\mathbf{Z}^{FOL}$-Entailment)
Let $\mathcal{KB}$ be a restricted first-order knowledge base, D be the domain and p a tolerance-pair for them. The corresponding ranking function κ_z^{FOL} induces a consequence relation in which a formula B can be z^{FOL}-entailed from A, denoted as $A \mid\!\sim_z^{FOL} B$, if and only if it is accepted by the model κ_z^{FOL}:

$$A \mid\!\sim_z^{FOL} B \text{ iff } \kappa_z^{FOL} \models (B|A)$$

In Chapter 5, properties of this consequence relation are analyzed and compared to the behavior of z-entailment in the propositional case. However, we can already see an essential difference to the propositional case: While κ_z in system Z is unique, κ_z^{FOL} depends on a specific tolerance-pair, of which we can have several ones for a knowledge base, each leading to a ranking function that is a model. The definition above should therefore be seen as a temporary one, which we want to refine in order to yield a unique consequence relation for a knowledge base.

3.3 Illustrating example

We now demonstrate system Z^{FOL} in a detailed example. The setting that has already been used in Example 2 to illustrate system Z is therefore transferred to a first-order language. The knowledge base is a rather simple one, as the focus here is on showing the technical details of the computation of κ_z^{FOL}. More interesting knowledge bases are analyzed later in this thesis.

Example 3 Our signature contains the predicates B, T, D and the domain $\{b, t\}$. Let our knowledge base $\mathcal{KB}_3$ consist of the following facts and conditionals:

$$\mathcal{KB}_3 = \langle\ \{B(b), T(t), \forall\, x\, T(x) \Rightarrow B(x)\}\ ,\ \{(D(x)|B(x)), (\overline{D}(x)|T(x))\}\ \rangle$$

Again, the first conditional says that bears are usually dangerous and the second conditional states that teddy bears are usually not dangerous. The certain knowledge expresses

that Bruno b is a bear and Teddy t is a teddy bear. Moreover, every teddy bear is also a bear, which means teddy bears are a subclass of bears, and due to the conditionals in $\mathcal{R}$, this subclass is an exceptional one with regard to being dangerous.

A possible tolerance-pair p for this knowledge base is

$$p = \langle\ (\{(D(x)|B(x))\}, \{b\})\ ,\ (\{(\overline{D}(x)|T(x))\}, \{t\})\ \rangle$$

which partitions the conditionals and the domain into two subsets. In order to check whether the tolerance-pair actually satisfies the conditions of Definition 16, we look at a specific world which shows that this is the case:

$$\omega_2 : B(b)B(t)\overline{T}(b)T(t)D(b)\overline{D}(t)$$

For $i = 0$, we have to test if $(D(b)|B(b))$ fulfills condition (1), (2) and (3). (1) obviously holds for ω_2, and since $B(b)D(b)$ is satisfied in ω_2, (2) holds as well. $(\overline{D}(b)|T(b))$ is not applicable in ω_2, hence (3) is also satisfied. For $i = 1$, (1) still holds in ω_2. (2) is satisfied because $(\overline{D}(t)|T(t))$ holds is ω_2 and (3) holds as there is no other conditional that could be falsified. Therefore, the partitions indeed define a tolerance-pair.

Based on this tolerance-pair, we can now compute the ranking function κ_z^{FOL}. For the world ω_2 the computation expands to the following formula:

$$\kappa_z^{FOL}(\omega_2) = \sum_{i=0}^{1} 3^i \lambda_i(\omega_2) - \kappa_0 = 1 \cdot \lambda_0(\omega_2) + 3 \cdot \lambda_1(\omega_2) - \kappa_0.$$

The final ranking is made up from the λ-values for each subset of the partition, while the factor $(m+2)^i$ ensures that the order of the constants is also taken into account. As we already discussed above, in ω_2 the conditional $(D(b)|B(b))$ is verified and $(\overline{D}(b)|T(b))$ is not applicable. This means, since no conditional is falsified, $\lambda_0(\omega_2)$ must be 0. For λ_1, we see that $(D(t)|B(t))$ is falsified and $(\overline{D}(t)|T(t))$ is verified. The assigned integer is one plus the highest subset label of the falsified conditionals, which is in this case 0, hence $\lambda_1(\omega_2) = 1$. Using these results, the formula above can be changed to

$$\kappa_z^{FOL}(\omega_2) = \sum_{i=0}^{1} 3^i \lambda_i(\omega_2) - \kappa_0 = 1 \cdot 0 + 3 \cdot 1 - \kappa_0 = 3 - \kappa_0.$$

Correspondingly, λ-values and rankings can be computed for all other possible worlds. In Figure 3.1, the results are listed. Worlds that are ranked ∞ because they do not satisfy the certain knowledge are not mentioned. The last unknown value, κ_0, is a normalization factor that is subtracted from every ranking to assure that κ_z^{FOL} is a proper ranking function with at least one world ranked with 0. In our example, κ_0 is 3, making ω_2 the most plausible world. The ranks shown in Figure 3.1 are already normalized.

Having computed the ranking function κ_z^{FOL}, we can now look at the induced consequence

	$B(b)$	$B(t)$	$T(b)$	$T(t)$	$D(b)$	$D(t)$	$\lambda_0(\omega)$	$\lambda_1(\omega)$	$\kappa(\omega)$
ω_0	1	1	0	1	0	0	1	1	1
ω_1	1	1	0	1	0	1	1	2	4
ω_2	1	1	0	1	1	0	0	1	0
ω_3	1	1	0	1	1	1	0	2	3
ω_4	1	1	1	1	0	0	1	1	1
ω_5	1	1	1	1	0	1	1	2	4
ω_6	1	1	1	1	1	0	2	1	2
ω_7	1	1	1	1	1	1	2	2	5

Figure 3.1: Ranking function κ_z^{FOL} of Example 3 for $\kappa_z^{FOL}(\omega) \neq \infty$

relation as given by Definition 18. To check whether we can conclude for a bear which is a teddy bear $B(x)T(x)$ that it is usually not dangerous $\overline{D}(x)$, the open conditional

$$(\overline{D}(x)|B(x)T(x))$$

has to be accepted by the ranking function. According to Definition 12, we first have to make sure that the conditional has at least one representative. Our bear b verifies the conditional in ω_4 and ω_5 with ranks 1 and 4, while our teddy bear t verifies it in ω_0, ω_2, ω_4 and ω_6, with 0 being the minimal rank. Therefore, t is the most plausible instance for this conditional. It also accepts the conditional, since it is minimally falsified in ω_3 with rank 3, and hence is the only representative.

Now, condition (Acc-1) requires the conditional to be more plausibly verified than falsified. As we already saw, the formula $B(x)T(x)\overline{D}(x)$ holds for t in ω_2 and is therefore ranked with 0. The falsification of the conditional, $B(x)T(x)D(x)$, minimally holds for b in ω_6, giving it a rank of 2. This means the condition (Acc-1)

$$\kappa(B(x)T(x)\overline{D}(x)) = 0 < 2 = \kappa(B(x)T(x)D(x))$$

holds and the conditional is accepted by κ_z^{FOL}. The conclusion $B(x)T(x) \mid\!\sim_z^{FOL} \overline{D}(x)$ can be made with z^{FOL}-entailment. ∎

3.4 Open questions

Having introduced system Z and system Z^{FOL}, a first comparison of the definitions and theorems points out the gaps that still have to be filled. A summary is given in Figure 3.2, with • indicating the mentioned gaps. We first notice that first-order knowledge bases allow a finer modeling, as certain knowledge can be explicitly given in addition to condi-

	System Z	System Z^{FOL}
Knowledge Base	Conditionals $\mathcal{R}$	Conditionals $\mathcal{R}$ and Facts $\mathcal{F}$
Tolerance & Partition	Partition of $\mathcal{R}$ and tolerance as in Definition 6	Tolerance-Pair Definition 16
Algorithm	Algorithm 1	*no algorithm yet* •
Consistency	Consistency **iff** partition Theorem 1	Consistency **if** Tolerance-Pair Theorem 3 (*also iff?* •)
Ranking Function	Definition 7	Definition 17
Properties	Minimality Theorem 2	*no unique with special properties* •
Consequence Relation	z-entailment Definition 8	z^{FOL}-entailment Definition 18
Benchmark Problems	Transitivity, Irrelevance, Specificity, no ex. Inheritance	*full evaluation to be done* •

Figure 3.2: Comparison of system Z and Z^{FOL} and open questions

tionals. The notion of tolerance and partitions of the knowledge base are a bit different, caring for the peculiarities of the languages, but exist in both cases.

The first major difference is that, while there is an algorithm to find a certain partition of a knowledge base in the propositional case, we do not yet have such an algorithm for the first-order case. Hence, providing an algorithm that finds tolerance-pairs seems to be a next step for the work on system Z^{FOL}. Further, we cannot prove a first-order knowledge base to be inconsistent with such an algorithm. If we find a tolerance-pair, we can construct a model and thereby show that the knowledge base is consistent, as Theorem 3 states. However, if we do not find a tolerance-pair, we do not yet know whether this means that the knowledge base actually has no model. In the propositional case, Theorem 1 provides such a guarantee.

Moreover, in both settings we can compute a ranking function from a partition or tolerance-pair. The partition obtained in the propositional case leads to a ranking function which is minimal. In the first-order case, in which several tolerance-pairs for a knowledge base can exist, we do not yet know which pair to choose to obtain a unique and suitable ranking function. Once there is a unique ranking function that can be computed for first-order knowledge bases, the next question is how the induced consequence relation handles the discussed benchmark examples. The remaining chapters of this thesis are dedicated to investigate these open questions of system Z^{FOL}.

4 Implementation

In order to analyze and evaluate the approach presented in the preceding chapter, system Z^{FOL}, a first implementation has been created as part of the work on this thesis. This chapter provides details about the implementation and shows how it can be used to investigate the new system. After introducing the objectives, a walk-through demonstrates the implementation, its user interface and the available functionality. Afterwards, the architecture and technical details of the program are briefly discussed.

4.1 Objectives and restrictions

Since system Z^{FOL} by Kern-Isbener and Beierle is a new approach that has recently been published, there is still work left that has to be done to fully understand the inner dynamics, advantages and limitations of the system. Open questions that have already been identified were presented in the previous section. Thus, the main objective for the work on the implementation was to provide a program that can be used to simplify the analysis of the system.

The program is implemented in Java and consists of data structures and algorithms realizing system Z^{FOL}, as defined in Chapter 3. As a basis, data structures for formulas and conditionals as well as a parser for knowledge bases from the library *Log4KR*, which is part of the *KReator* toolbox, are used.[1] In addition, a simple graphical user interface on top of the implemented data structures and algorithms is provided to interact with knowledge bases, tolerance-pairs and ranking functions in a convenient way.

The overall objective, which is to provide a useful tool that simplifies the study and evaluation of the new approach, can be broken down into more concrete requirements:

Transparency In order to understand the dynamics of system Z^{FOL}, the program has to provide as much details about its actions and calculations as needed. Therefore, it offers a detail-mode for certain actions. For instance, if we want to evaluate whether a given conditional is accepted by a ranking function or not, switching to the detail-mode makes

[1] KReator, a toolbox for relational probabilistic knowledge representation, reasoning and learning, is part of the KReate project, a joint project of TU Dortmund, Department of Computer Science, Chair VI (Information Engineering) and FernUni in Hagen, Department of Computer Science, Chair VIII (Wissensbasierte Systeme). For more information, see `http://kreator-ide.sourceforge.net/`

the program output detailed information about every step of the evaluation with reference to the corresponding definitions.

Usability and Reproducibility The program should be easy to use and the results obtained should be reproducible. To satisfy these requirements, a graphical user interface as well as a file interface are offered. Using the latter, we can import and export knowledge bases and save every output of the program to a file.

Modularity and Extensibility The basic data structures and algorithms should be provided in a way in which they can be extended and reused in other contexts. Following the approach of library *Log4KR*, this is achieved by consequently applying an object-oriented design, making use of inheritance, abstraction, generics and interfaces.

Two general limitations of the implementation are known: First, due to restrictions of the parser from library *Log4KR*, the program cannot read formulas containing implications and quantifications. Only negations, conjunctions and disjunctions are supported. A detailed explanation of the expected syntax follows in the next section. Second, as the main focus is on providing a transparent implementation of system Z^{FOL}, most algorithms are naive implementations of the corresponding definitions. Therefore, for larger knowledge bases, the runtime can become very high. However, though more advanced algorithms could improve the runtime to some extent, this is also a problem of system Z^{FOL} in general and not of this specific implementation.

The program has been tested with a Java 7.0 Standard Edition runtime environment. It uses library *Log4KR* as of revision 3811. With this thesis, a compiled and executable version of the program, also containing the library, together with sources and a Javadoc-documentation are provided. In Appendix A.3 on page 83, a detailed list of the provided artifacts is available. The program is presented in the next two sections. While the first one shows it from a user's perspective, the second one gives the technical details.

4.2 User interface walk-through

This section demonstrates and explains the features of the implemented program by walking through each part of the user interface. It can be seen as the manual of the program. During the demonstration, a new example knowledge base for system Z^{FOL} is introduced that is used, in different variations, throughout the rest of the thesis.

Figure 4.1 shows the user interface. It consists of three main parts: a tool bar on the top and an input and an output area in the bottom left and right. If we want to provide an input to the program, for instance, a knowledge base or a query conditional, we can use the input area. Every output produced by the program is written to the output area. At any time, we can use the *Open* and *Save* functions of the areas to read from or write to

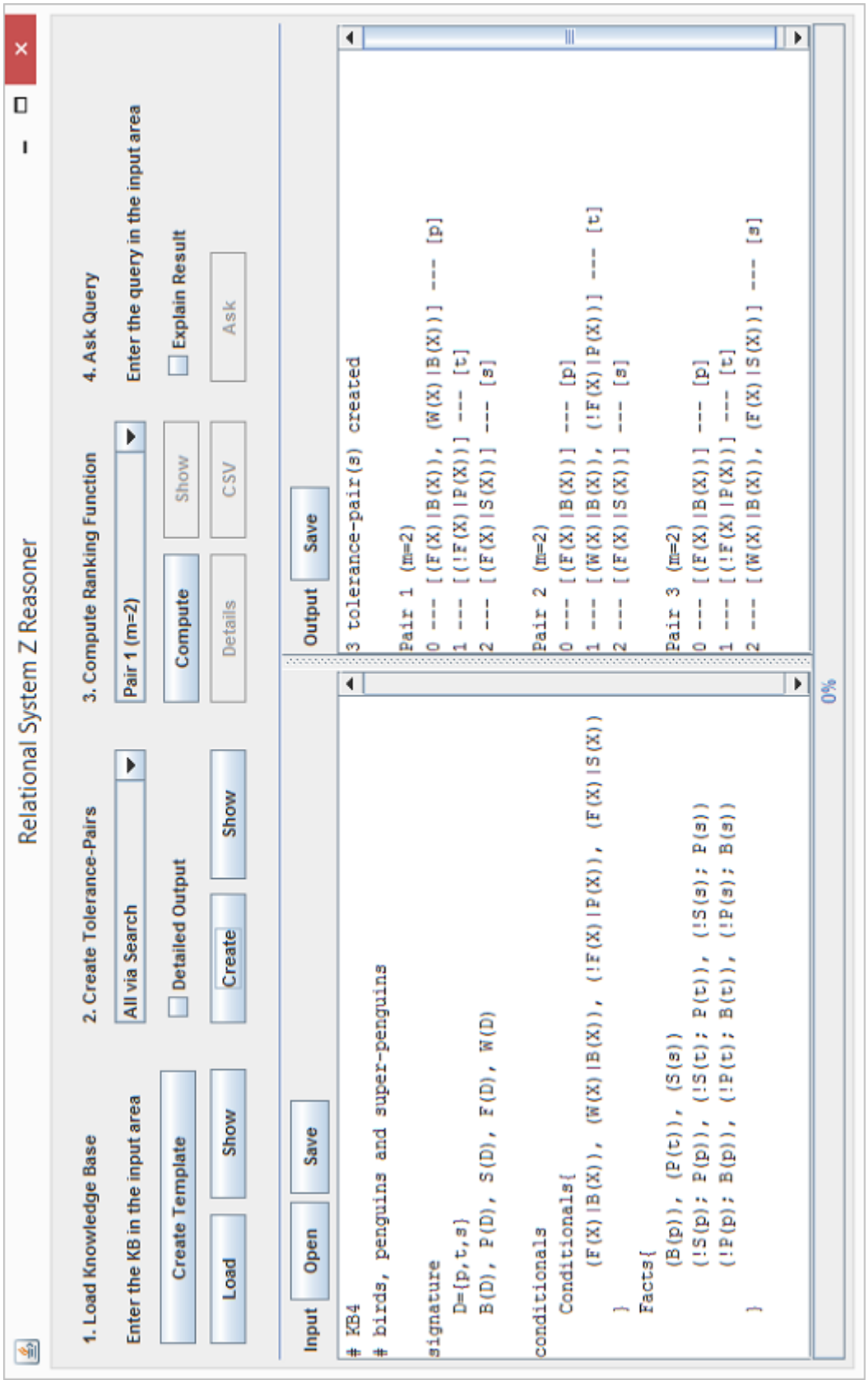


Figure 4.1: User interface of the implemented program

a file. By sliding the separator between the areas to the left or right, the space occupied by the areas can be adjusted.

The tool bar functions are structured along a four-step process that we will also follow in this demonstration. First of all, in *step 1*, a knowledge base has to be loaded. Since it is expected to be in the input area, we can either import it from a file via *Open* or type it directly. The following functions are offered:

Load	Parses the given input and writes the loaded knowledge base to the output area. If errors occur, they are written to the output area instead.
Show	Writes the knowledge base currently loaded to the output area.
Create Template	Provides a short example of a knowledge base in the input area as a starting point to create a new knowledge base.

The expected syntax for a knowledge base is now illustrated by an example. We introduce one of the most popular examples in non-monotonic reasoning, which is known as the *bird-penguin-scenario* or the *Tweety-example*. A system Z^{FOL}-specific version of it, taken from [KIB15], is used here. It is the running example for the rest of this thesis.

Example 4 We want to load the following knowledge base:

$$\begin{aligned}\mathcal{KB}_4 = \langle\ &\{\forall x P(x) \Rightarrow B(x), \forall x S(x) \Rightarrow P(x), B(p), P(t), S(s)\} \\ &\{(F(x)|B(x)), (W(x)|B(x)), (\overline{F}(x)|P(x)), (F(x)|S(x))\}\ \rangle\end{aligned}$$

The domain should be $D = \{p, t, s\}$, consisting of three individuals, usually known as Polly (p), Tweety (t) and Supertweety (s). The facts of $\mathcal{KB}_4$ describe that Polly is a bird (B), Tweety a penguin (P) and Supertweety a super-penguin (S). In addition, they state that all super-penguins are penguins and all penguins are birds, giving us a two-level hierarchy of classes of birds, with Supertweety being the most specific one. The conditionals define the ability to fly (F) for each of the classes: While birds usually fly, penguins usually cannot fly. However, super-penguins are an exception, they do usually fly. And, finally, birds usually have wings (W), and there is no exception for subclasses regarding this rule.

In order to load this knowledge base into the program, we have to provide the domain, predicates, conditionals and facts in the following way in the input area:

```
signature
   D={p,t,s}
   B(D), P(D), S(D), F(D), W(D)

conditionals
```

```
Conditionals{
   (F(X)|B(X))
   (W(X)|B(X))
   (!F(X)|P(X))
   (F(X)|S(X))
}
Facts{
   (B(p)), (P(t)), (S(s))
   (!S(p); P(p)), (!S(t); P(t)), (!S(s); P(s))
   (!P(p); B(p)), (!P(t); B(t)), (!P(s); B(s))
}
```

Once the knowledge base has been successfully loaded, the program prints the parsed contents into the output area:

```
Current knowledge base:

Predicates
[B(D), P(D), S(D), F(D), W(D)]

Domain
p, t, s

Conditionals
[(F(X)|B(X)), (W(X)|B(X)), (!F(X)|P(X)), (F(X)|S(X))]

Facts
[B(p), P(t), S(s), (!S(p) ; P(p)), (!S(t) ; P(t)), (!S(s) ; P(s)),
(!P(p) ; B(p)), (!P(t) ; B(t)), (!P(s) ; B(s))]

Possible Worlds: 32768
```

In addition to listing the predicates, constants, conditionals and facts, the number of possible worlds for this knowledge base is also printed. ■

The example showed certain limitations of the parser that were already mentioned in the previous section. Only negation (`!`), conjunction (`,`) and disjunction (`;`) can be used in formulas. Material implications need to be expressed by negation and disjunction. Quantifications cannot be given directly, they have to be transformed into conjunctions or disjunctions of formulas for each individual, which is always possible because of the finite domain. The example shows this way of handling quantifications for the two facts. Facts and conditionals have to be surrounded by parentheses, variables must start with

an upper-case letter. In Appendix A.1 on page 77, a complete and precise definition of the expected syntax is available, which also gives some background on why the syntax has to be that way. The restrictions arise only because an already implemented parser was used which could not be extended, but of course, in the future, it would be possible to overcome them by implementing a specific parser for this program.

Once the knowledge base is loaded, we can proceed with *step 2*, in which the program creates tolerance-pairs for the knowledge base. Here, three different approaches are implemented:

1. *All via Brute Force*
2. *All via Search*
3. *Minimal via Search*

While the first two options create every possible tolerance-pair for a knowledge base, the last one only creates a subset. In Section 5.2 and Section 5.3, the different approaches and the underlying algorithms are explained and discussed in depth. Besides choosing one of the three methods, the following functions are available:

Create	Creates tolerance-pairs using the chosen method. For the first method, the progress is indicated at the bottom of the user interface. When the search finishes, all created tolerance-pairs are written to the output area. A search run can be canceled using the same button.
Show	Prints the tolerance-pairs found during the last run to the output area.
Detailed Output	If this option is selected, tolerance-pairs are printed together with a world and a constant for each conditional as required in Definition 16.

Using the last option, we can double-check that a created tolerance-pair is in fact a valid tolerance-pair. We continue the example and execute step 2:

Example 4 (continued) We select the first method and press *Create*. The following result is written to the output area:

```
3 tolerance-pair(s) created

Pair 1 (m=2)
0 --- [(W(X)|B(X)), (F(X)|B(X))] --- [p]
1 --- [(!F(X)|P(X))] --- [t]
2 --- [(F(X)|S(X))] --- [s]

Pair 2 (m=2)
0 --- [(F(X)|B(X))] --- [p]
```

```
1 --- [(!F(X)|P(X)), (W(X)|B(X))] --- [t]
2 --- [(F(X)|S(X))] --- [s]

Pair 3 (m=2)
0 --- [(F(X)|B(X))] --- [p]
1 --- [(!F(X)|P(X))] --- [t]
2 --- [(W(X)|B(X)), (F(X)|S(X))] --- [s]
```

Hence, for our knowledge base $\mathcal{KB}_4$ and domain $D = \{p, t, s\}$, three tolerance-pairs exist. The only difference between them is that $(W(x)|B(X))$ is placed in another subset. ∎

Now, the drop down list of *step 3* is filled with the created tolerance-pairs. By choosing one of them, we can proceed with the computation of the corresponding ranking function. In this step, the following functions are available:

Compute Computes κ_z^{FOL} based on the selected tolerance-pair as defined in Definition 17. As a result, all worlds not ranked ∞ a printed with their ranking, sorted by plausibility, to the output area.

Show The output of the last computation is written to the output area again.

Details A popup dialog prompts the user to choose one of the worlds satisfying the facts. Then, a detailed computation of the rank for this world is shown. The details are calculated ad-hoc and not stored during the initial *Compute*-run.

CSV The ranks for all possible worlds, sorted by truth assignments, are printed. This output is optimized to be imported into a spread sheet application, with columns delimited by a semicolon. Rank ∞ is represented as `inf`.

If a knowledge base has several possible tolerance-pairs, the resulting ranking functions can be easily compared by exporting both into a spread sheet application using function *CSV*. An corresponding example is given in the next chapter. Here, in order to continue the running example, we will focus on the *Detail* function:

Example 4 (continued) We compute the ranking function for the first tolerance-pair and request a detailed output for the first world offered in the popup. The following result is written to the output area:

```
Selected tolerance-pair:
Pair 1 (m=2)
0 --- [(F(X)|B(X)), (W(X)|B(X))] --- [p]
1 --- [(!F(X)|P(X))] --- [t]
2 --- [(F(X)|S(X))] --- [s]
```

```
World:
w = (B(p)=1 B(t)=1 B(s)=1 P(p)=0 P(t)=1 P(s)=1 S(p)=0 S(t)=0 S(s)=1
     F(p)=0 F(t)=0 F(s)=0 W(p)=0 W(t)=0 W(s)=0)

Satisfies facts: true
Ranking: 17

lambda(0,w) = 1
lambda(1,w) = 1
lambda(2,w) = 3

k_0: 36

k(w) = 1 * 1 + 4 * 1 + 16 * 3 - 36 = 17
```

We can see the tolerance-pair and the selected world. It satisfies the facts and is ranked with 17. The values λ_i for each $i \in \{0, \ldots, m\}$ as well as κ_0 are shown, making the computation in the last line comprehensible. ■

Finally, after a ranking function has been computed based on a tolerance-pair, in the last *step 4*, we can evaluate queries against it. If a formula or a conditional is provided in the input area[2], pressing *Ask* checks whether it is accepted by the ranking function according to Definition 12. By testing the acceptance of conditionals, we can also investigate the reasoning behavior of z^{FOL}-entailment, which was defined in Definition 18. In the standard mode, the program only returns `true` or `false` to indicate the acceptance of the query. However, by activating the option *Explain Result*, we can make the program show us the whole evaluation process. The next example illustrates this option.

Example 4 (continued) We continue with the ranking function we computed before, based on the first tolerance-pair. First, we want to test whether our bird Polly usually flies. The corresponding conditional is `(F(p)|B(p))`. In the *Explain Result* mode, the program returns the following output:

```
0: Acceptance by ranking function k
  1: k |= (F(p)|B(p)) -> true
    2: k( B(p) F(p) ) = 0 (Def. 3)
      3: (B(p)=1 B(t)=1 B(s)=1 P(p)=0 P(t)=1 P(s)=1 S(p)=0 S(t)=0 S(s)=1
          F(p)=1 F(t)=0 F(s)=1 W(p)=1 W(t)=0 W(s)=0)
```

[2] Based on the EBNF in Appendix A.1 on page 77, a formula must comply with the rule `formula` and a conditional with `conditional`. Only one formula or conditional can be provided at a time.

```
2: k( B(p) !F(p) ) = 1 (Def. 3)
  3: (B(p)=1 B(t)=1 B(s)=1 P(p)=0 P(t)=1 P(s)=1 S(p)=0 S(t)=0 S(s)=1
      F(p)=0 F(t)=0 F(s)=1 W(p)=0 W(t)=0 W(s)=0)
2: 0 < 1 ? (Def. 4)
```

We can see that the conditional is accepted. First, for both the verifying formula $B(p)F(p)$ and the falsifying formula $B(p)\overline{F}(p)$, the ranks are derived by finding the most plausible world satisfying the formula. After that, by comparing both ranks, the acceptance can be determined. The program provides the full evaluation tree down to the most basic definition and directly refers to the relevant ones by stating their numbers. Please note that the actual program refers to the definitions as numbered in [KIB15], whereas the output shown here has been modified to fit to the numbering used in this thesis.

As a second example, we evaluate whether Tweety has wings. Using the formula `W(t)` as a query yields the following result:

```
0: Acceptance by ranking function k
  1: k |= W(t) -> false
    2: W(t) invalid in world with k=0 (Def. 4)
    2: (B(p)=1 B(t)=1 B(s)=1 P(p)=0 P(t)=1 P(s)=1 S(p)=0 S(t)=0 S(s)=1
        F(p)=1 F(t)=0 F(s)=1 W(p)=1 W(t)=0 W(s)=0)
```

A formula is only accepted if it is satisfied in every world ranked most plausible, that is, all worlds with rank 0. However, our formula $W(t)$ is falsified in one of these worlds, which is given in the output. Therefore, it is not accepted. ■

In contrast to the output shown here, the evaluation tree is shown as an interactive tree in the program. Instead of the output area, a tree control is available in this case in which branches of the tree can be individually expanded and collapsed by the user to explore the evaluation result. When the *Save*-function is used to store the result in a file, the tree is converted into the textual representation that is shown in the example above. With this usability feature, even complex evaluations such as for open conditionals can easily be followed and investigated.

Based on this implementation, certain analyses and observations regarding system Z^{FOL} were made that are presented in Chapter 5. For the sake of legibility, knowledge bases introduced during the remainder of this thesis are given in the concise mathematical notation already used before. In addition, in Appendix A.2 on page 78, an import-ready version of each knowledge base is provided that can be used with the program.

4.3 Technical architecture and design

Based on the *Log4KR* library, system Z^{FOL} is implemented by a set of classes realizing the necessary data structures and algorithms. The user interface is built with Java Swing. This section should give an overview of the implementation and its general architecture, while all details can be found in the Javadoc-documentation.

From *Log4KR*, data structures for formulas and their interpretation are used. The library makes use of generics to model both propositional and first-order logic in a common framework. An atom, modeled by the interface *Interpretable*, is the basic building block for a formula. It is implemented by classes *PropositionalVariable* for the propositional and by *RelationalAtom* for the first-order case. Complex formulas can be built with corresponding classes which always implement the interface *Formula<I>*, with I being a type of *Interpretable*. The generic class *Conditional<I>* combines two formulas as antecedence and consequence, modeling a conditional, while there is also an inherited version for the first-order case, *RelationalConditional*. Objects of these structures can be generated from textual representations using the parser *Log4KRReader*, which is an essential part of the created program. Regarding the semantics, the library provides an abstract class *Interpretation<I>* that offers methods to test whether formulas (of type I) are satisfied or not. The derived class *PossibleWorldMapRepresentation<I>* assigns truth values to the interpretables of a language and thereby represents a possible world. Class *PossibleWorldFactory<I>* systematically generates all possible assignments. Based on these classes, the program is built.

In Figure 4.2, the most important classes of the implementation are shown in an UML class diagram. *RankingFunction<I>* realizes a mapping from possible worlds, that can be created as mentioned above, to integers. The methods *getRank* and *accepts* implemented Definition 3 and Definition 4. Following the approach of the library, the class is generic in order to support both propositional and first-order formulas. While this implementation covers the full definition of a ranking function in the propositional case, the inherited class *RelationalRankingFunction*[3] provides additional methods for the first-order case. It can handle open formulas and conditionals as in Definition 11 and Definition 12 and determine representatives as in Definition 13, but delegates calls for closed formulas and conditionals to its parent class.

To implement system Z^{FOL}, two additional classes realize necessary data structures. A *RelationalKnowledgeBase* consists of a set of first-order conditionals and a set of first-order formulas and represents a first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$. Objects of class *TolerancePair* store a partition of both conditionals and constants into *nOfParts* subsets

[3] Following the naming concept of *Log4KR*, classes dealing with first-order language are prefixed with *relational*. In this thesis, however, we always use the term *first-order* when referring to the defined subset of first-order logic that is relevant for system Z^{FOL}.

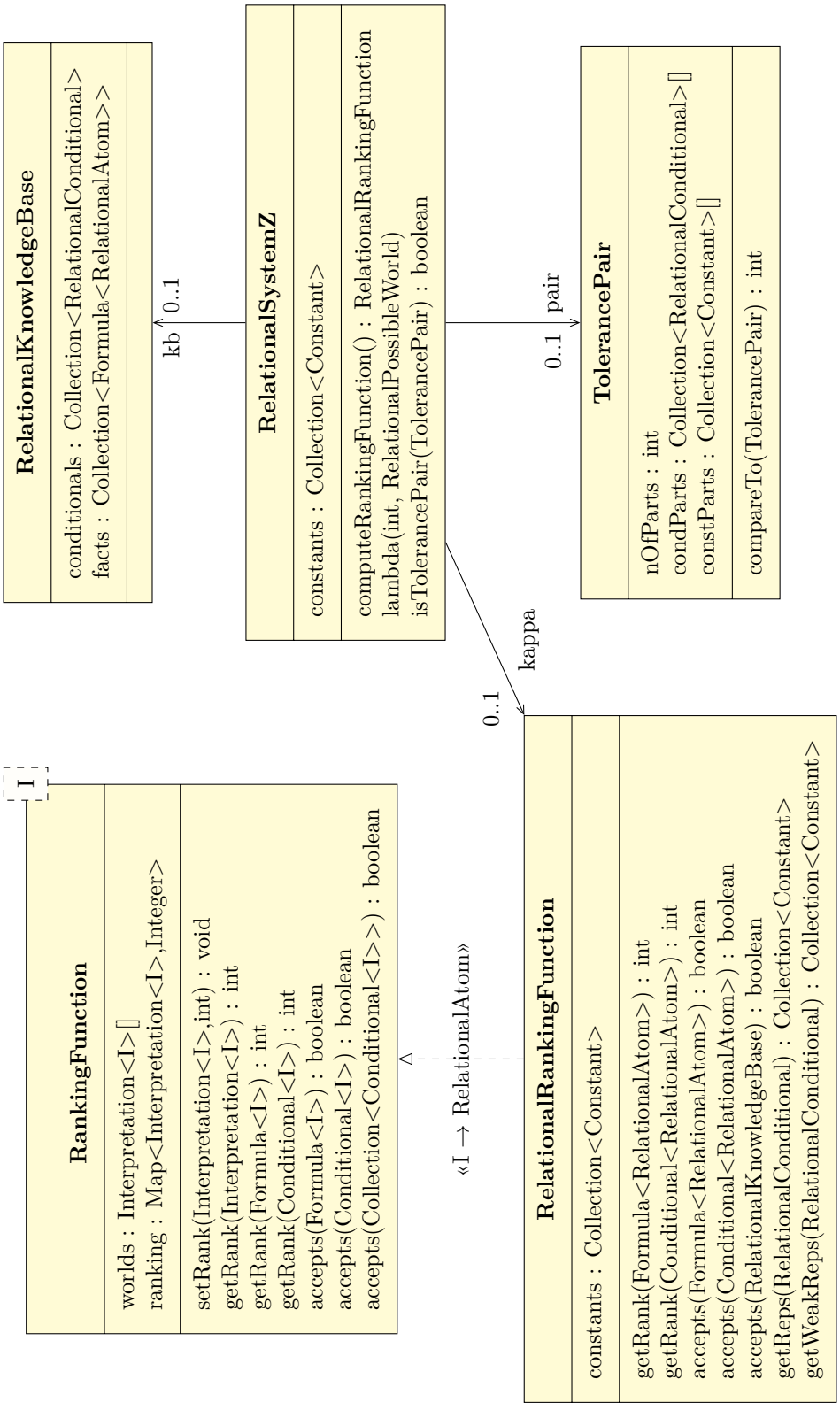


Figure 4.2: Core classes of the implementation

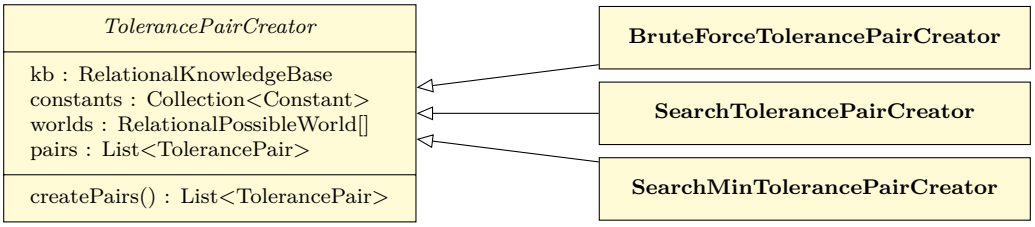


Figure 4.3: Implementation of tolerance-pair creators

as defined in Definition 16. However, the data structure itself does not check the conditions but can store any partition. In *RelationSystemZ*, the actual check is implemented in method *isTolerancePair*. An instance of *RelationSystemZ* holds a first-order knowledge base, a domain and a tolerance-pair and can compute the ranking function κ_z^{FOL} according to Definition 17. The corresponding logic is implemented in *computeRankingFunction* and *lambda*. With theses classes, the core functionality of system Z^{FOL} is realized.

Another interesting aspect of the program's technical design is how the different approaches to find tolerance-pairs are implemented. An abstract class *TolerancePairCreator* defines the general contract for such an approach. It has a first-order knowledge base, a domain and the corresponding possible worlds and offers a method that creates and returns tolerance-pairs based on the information. Currently, three concrete classes implement the different approaches mentioned in the previous section as depicted in Figure 4.3. Due to this flexible design, additional methods can be easily added later.

Additional classes realize the user interface and the generation of the detailed explanation offered by the program. However, they should not be discussed in detail here. A full technical documentation is provided with the program and covers all objects.

The implemented algorithms in the program are mostly direct translations of the corresponding definitions into code. That means, for instance, that a rank for a formula is calculated by iterating over all worlds of the ranking function and searching for the minimal rank of the worlds that satisfy the formula. More complex evaluations such as the acceptance of an open conditional are implemented using the more basic evaluations. As no intermediate results are buffered, this can lead to numerous iterations over the set of possible worlds in order to obtain ranks and determine acceptance.

As an example, we look at the algorithm used to check a potential tolerance-pair for validity. The corresponding pseudo code is given as algorithm IsTolerancePair and is implemented in class *RelationalSystemZ*. Given a partition of conditionals and constants as well as a set of facts, it checks whether they fulfill the conditions specified in Definition 16. The algorithm returns a positive result if the provided tolerance-pair is indeed a tolerance-pair for the first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and domain D, with $\mathcal{R}$ and D made up as the union of the conditional and constants in the partitions. For each conditional (line 1), it tries to find a world that satisfies the facts of the knowledge base (line 3). In

Algorithm 2 IsTolerancePair

Input: potential tolerance-pair p and facts $\mathcal{F}$

Output: `true`, if p is a tolerance-pair for $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ with $\mathcal{R} = p_{\mathcal{R},0} \cup \ldots \cup p_{\mathcal{R},m}$ and $D = p_{D,0} \cup \ldots \cup p_{D,m}$, otherwise `false`

1: **for** $r \in p_{\mathcal{R},i}, 0 \leq i \leq m$ **do**
2: $hasWorld \leftarrow$ `false`
3: **for** $\omega \in \Omega$ with $\omega \models \mathcal{F}$ **do**
4: **for** $a \in p_{D,i}$ **do**
5: **if** ω verifies $r(a)$ and not falsifies $r'(a')$ for all $r' \in \cup_{j \geq i} p_{\mathcal{R},j}, a' \in p_{D,i}$ **then**
6: $hasWorld \leftarrow$ `true`
7: **if not** $hasWorld$ **then**
8: **return** `false`
9: **return** `true`

this world, the conditional must be verified (line 5) instantiated with a constant from the corresponding subset of D (line 4). Further, no conditional from the same or following subsets of $\mathcal{R}$ might be falsified instantiated with any constant from the corresponding subset of D (line 5). If such a world can be found for every conditional, the pair is in fact a tolerance-pair in the sense of Definition 16.

Since both the set of predicates $\mathcal{P}$ and the domain D are finite and only unary predicates are allowed, there are $|\mathcal{P}| \cdot |D|$ possible atoms and hence $2^{|\mathcal{P}||D|}$ possible worlds. For a given world, we can check in linear time, dependent on the number of atoms, whether a formula is satisfied or not. In the worst-case scenario, algorithm IsTolerancePair runs for $|R|$ conditionals, each time over $2^{|\mathcal{P}||D|}$ worlds and for $|D|$ constants and, in the innermost loop, has to check $1 + |R| \cdot |D|$ conditionals for verification or falsification. Clearly, the dominating factor is the iteration over possible worlds and the algorithm therefore has exponential time complexity $\mathcal{O}(2^n)$, with $n = |\mathcal{P}| \cdot |D|$.

In general, the following remarks can be made regarding the runtime of the program: As almost every implemented algorithm iterates over possible worlds, the numbers of predicates and constants are crucial for the programs runtime. In step 1, when a knowledge base is loaded, possible worlds are generated, for which also sufficient memory has to be available. The runtime of the algorithms of step 2 is analyzed in the next chapter. The calculation of the ranking function in step 3 requires two iterations over the set of possible worlds and is therefore relatively fast compared to the other steps. Evaluating a query, in step 4, can consist of multiple iterations over the worlds, depending on the structure of the query. Although the runtime might disqualify the program for practical applications, it should be usable for the evaluation of system Z^{FOL}, which is the overall objective guiding the implementation as stated in the first section of this chapter.

5 Analysis

In this chapter, the findings of the analysis of system Z^{FOL} that have been obtained using the implementation are presented. The relation between the new system and the classical system Z is investigated first. Then, conditions for the existence of tolerance-pairs for a knowledge base and different approaches to find them are studied. Finally, the reasoning behavior of z^{FOL}-entailment and first-order properties of the new approach are analyzed. The key findings, which are partly of a formal and partly of an intuitive nature, are stated as *observations*. Throughout the chapter, different variations of the bird-penguin-scenario introduced in the previous chapter are used to illustrate the findings.

5.1 Propositional edge case

A first comparison of the corresponding definitions and theorems in Section 3.4 outlined similarities and open questions. Based on the fact that propositional logic is an edge case of first-order logic, this section continues the comparison of the two systems and analyzes their relation. Given a first-order knowledge base built only with predicates of arity zero, which means we essentially deal with a propositional language, the question is whether the application of system Z^{FOL} and system Z to it yield the same results.

In order to investigate this question, we need to make a few modifications to the definition of a first-order knowledge base for system Z^{FOL}. Temporarily, we require all predicates to be of arity zero. That means that we do not deal with variables and quantifications in formulas and conditionals, do not need constants for instantiating open formulas and an interpretation is merely a truth-assignment to predicates. Our language then corresponds to a propositional conditional language. Also, we require our knowledge base to have only conditional knowledge and no facts.

The propositional edge case of a first-order knowledge base and system Z^{FOL} is illustrated by an example that is the propositional counterpart of the bird-penguin-scenario. The knowledge base can be analyzed using the program presented in the previous chapter, as it can also handle knowledge bases of the edge case type described above.

Example 5 The propositional and first-order knowledge bases are defined as follows:

$$\mathcal{KB}_5^{PL} = \{(F|B), (W|B), (\bar{F}|P), (B|P), (F|S), (P|S)\}$$
$$\mathcal{KB}_5^{FOL} = \langle \emptyset, \mathcal{KB}_5^{PL} \rangle$$

While $\mathcal{KB}_5^{PL}$ is a propositional knowledge base, $\mathcal{KB}_5^{FOL}$ is a first-order knowledge base satisfying the new requirements made before. It represents the same knowledge as the propositional knowledge base. For both, the same possible worlds exist. ∎

If algorithm ConsistencyCheck is applied to $\mathcal{KB}_5^{PL}$, it finds a partition dividing the conditionals into three subsets:

$$\mathcal{R}_0 = \{(F|B), (W|B)\}, \quad \mathcal{R}_1 = \{(\bar{F}|P), (B|P)\}, \quad \mathcal{R}_2 = \{(F|S), (P|S)\}$$

For our edge case first-order knowledge base, the notion of a tolerance-pair is simpler than in the general case. Since we do not deal with constants, it is only a partition of the conditionals, exactly as in the case of system Z. The conditions of Definition 16 can be simplified as well, as the existence of $a \in D_i$ is not necessary to check the verification and falsification in a possible world. Condition (1) does not apply because we do not deal with facts in our edge case analysis. Conditions (2) and (3) then define a partition in which every conditional is tolerated by the conditionals in $\cup_{i \geq j} \mathcal{R}_j$, and that is exactly what ConsistencyCheck creates for system Z.

Observation 1 *In the propositional edge case of system Z^{FOL}, the notion of a tolerance-pair coincidences with a partition in system Z. In both cases, each conditional in the partition is tolerated by the conditionals in the same or higher subsets.*

The partition given above is therefore also a tolerance-pair for $\mathcal{KB}_5^{FOL}$. So far, system Z^{FOL} indeed turns out to be a direct extension of system Z to the first-order case.

Next, we compare the ranking functions created by system Z and system Z^{FOL}. Figure 5.1 shows the ranks of κ_z and κ_z^{FOL} for all possible worlds, computed according to Definition 7 and Definition 17. For the latter, again, instantiations of the conditionals can be ignored in the definition, as they are already grounded. This means the value of the resulting simplified version of λ of Definition 17 does no longer depend on the value of i:

$$\forall\, \omega \in \Omega : \forall\, (i,j) \in \{0,\ldots,m\} \times \{0,\ldots,m\} : \;\; \lambda_i(\omega) = \lambda_j(\omega) \tag{5.1}$$

In addition, λ is then equivalent to κ_z, because the definition of λ in the edge case is the same as Definition 7 for κ_z. For an arbitrary $i \in \{0,\ldots,m\}$, both functions assign the same value to a world:

$$\forall\, \omega \in \Omega : \;\; \lambda_i(\omega) = \kappa_z(\omega) \tag{5.2}$$

In Figure 5.1 we can see this equivalence of λ and κ_z for our example. Moreover, since we required the set of facts to be empty in an edge case first-order knowledge base, there is no world ranked with ∞ in κ_z^{FOL}, which is the case in κ_z anyway.

The rankings itself are not equal, but in the example we can see a relation between the

	B	P	S	F	W	$k_z(\omega)$	$k_z^{FOL}(\omega)$	$\lambda_0(\omega)$	$\lambda_1(\omega)$	$\lambda_2(\omega)$
ω_0	0	0	0	0	0	0	0	0	0	0
ω_1	0	0	0	0	1	0	0	0	0	0
ω_2	0	0	0	1	0	0	0	0	0	0
ω_3	0	0	0	1	1	0	0	0	0	0
ω_4	0	0	1	0	0	3	63	3	3	3
ω_5	0	0	1	0	1	3	63	3	3	3
ω_6	0	0	1	1	0	3	63	3	3	3
ω_7	0	0	1	1	1	3	63	3	3	3
ω_8	0	1	0	0	0	2	42	2	2	2
ω_9	0	1	0	0	1	2	42	2	2	2
ω_{10}	0	1	0	1	0	2	42	2	2	2
ω_{11}	0	1	0	1	1	2	42	2	2	2
ω_{12}	0	1	1	0	0	3	63	3	3	3
ω_{13}	0	1	1	0	1	3	63	3	3	3
ω_{14}	0	1	1	1	0	2	42	2	2	2
ω_{15}	0	1	1	1	1	2	42	2	2	2
ω_{16}	1	0	0	0	0	1	21	1	1	1
ω_{17}	1	0	0	0	1	1	21	1	1	1
ω_{18}	1	0	0	1	0	1	21	1	1	1
ω_{19}	1	0	0	1	1	0	0	0	0	0
ω_{20}	1	0	1	0	0	3	63	3	3	3
ω_{21}	1	0	1	0	1	3	63	3	3	3
ω_{22}	1	0	1	1	0	3	63	3	3	3
ω_{23}	1	0	1	1	1	3	63	3	3	3
ω_{24}	1	1	0	0	0	1	21	1	1	1
ω_{25}	1	1	0	0	1	1	21	1	1	1
ω_{26}	1	1	0	1	0	2	42	2	2	2
ω_{27}	1	1	0	1	1	2	42	2	2	2
ω_{28}	1	1	1	0	0	3	63	3	3	3
ω_{29}	1	1	1	0	1	3	63	3	3	3
ω_{30}	1	1	1	1	0	2	42	2	2	2
ω_{31}	1	1	1	1	1	2	42	2	2	2

Figure 5.1: Ranking functions κ_z and κ_z^{FOL} of Example 5

values. Based on the definitions of their computation, we can derive the connection, which turns out to be a factor that depends on the number of subsets in the partition.

Observation 2 *In the propositional edge case of system Z^{FOL}, the rank assigned to a possible world $\omega \in \Omega$ differs from the rank in system Z only by a factor f:*

$$\kappa_z^{FOL}(\omega) = f \cdot \kappa_z(\omega) \quad \textit{with} \quad f = \frac{(m+2)^{m+1} - 1}{m+1} \tag{5.3}$$

Proof Let $\omega \in \Omega$ be a possible world. According to Definition 17, the rank for ω is

$$\kappa_z^{FOL}(\omega) = \sum_{i=0}^{m} (m+2)^i \lambda_i(\omega) - \kappa_0 \quad \text{with} \quad \kappa_0 = \min_{\omega \in \Omega} \sum_{i=0}^{m} (m+2)^i \lambda_i(\omega).$$

Due to (5.1), we can extract the part that is independent of i from the summation.

$$\kappa_z^{FOL}(\omega) = \lambda_i(\omega) \cdot \sum_{i=0}^{m} (m+2)^i - \kappa_0 \tag{5.4}$$

Further, applying (5.2), we can replace λ by κ_z, which yields

$$\kappa_z^{FOL}(\omega) = \kappa_z(\omega) \cdot \sum_{i=0}^{m} (m+2)^i - \kappa_0. \tag{5.5}$$

If algorithm ConsistencyCheck found a partition in system Z, there must be at least one conditional in $\mathcal{R}_0$ for which a world exists in which it is verified and no other conditional is falsified. Hence, κ_z must assign 0 to at least one world. To this world, κ_z^{FOL} also assigns 0 because of (5.5), making $k_0 = 0$. Further, the summation of increasing powers of a number x can be simplified to a fraction (cf. [MS08, p. 270]):

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1} \text{ for } x \neq 1 \tag{5.6}$$

With both $k_0 = 0$ and (5.6), we can transform (5.5) to (5.3). ∎

In our example, the partition has three subsets ($m = 2$) and the factor f can be calculated with (5.3) to $f = (4^3 - 1)/3 = 63/3 = 21$. For the world ω_{30}, the rank $\kappa_z^{FOL}(\omega_{30})$ is then calculated as $\kappa_z^{FOL}(\omega_{30}) = 21 \cdot \kappa_z(\omega_{30}) = 21 \cdot 2 = 42$. Figure 5.1 shows that this relation holds for every possible world in our example.

The last observation also shows that system Z^{FOL} does not necessarily create a minimal ranking function, as it is the case for system Z. But since the factor f is constant for all worlds, we know that both ranking functions accept the same conditionals.

Observation 3 *In the propositional edge case of system Z^{FOL}, a conditional is accepted by κ_z^{FOL} if and only if it is accepted by κ_z.*

Proof We do not have to deal with quantified and open conditionals in the edge case. In Definition 4, acceptance is defined based on the $<$-relation on the set of integers. If the rank of each world is multiplied with a constant factor, their $<$-relations stay the same and hence both acceptance relations yield the same results. ∎

All in all, we can conclude that the propositional edge case of system Z^{FOL} in fact coincidences with system Z as expected. The notion of tolerance becomes the same and rankings can be directly transformed into each other, though they are not exactly the same. The slight difference is due to the features added in system Z^{FOL} that ensure the partition of the domain is also incorporated into the ranking. However, on a semantical level, both ranking functions are equivalent. In the following sections, system Z^{FOL} is further analyzed, now based on the original definitions given in Chapter 3.

5.2 Existence of tolerance-pairs

In the general case of system Z^{FOL}, the full characterization of a tolerance-pair as in Definition 16 applies, requiring a partition of both the conditionals and the domain that satisfy the defined conditions. This section investigates under which circumstances such a tolerance-pair exists. Different changes to a knowledge base and their influence on the existence of tolerance-pairs are illustrated by examples. First, we take a closer look at the term partition and formally define it. So far, every use of the term in this thesis implicitly assumed the partition to be ordered. While this property is also formally given in the definition, we will continue to implicitly assume this. In order to distinguish between a proper tolerance-pair and a pair of partitions that does not necessarily satisfy the conditions of a tolerance-pair, we finally introduce the term *partition-pair*.

Definition 19 (Partition(-Pair))
A (k-)partition of a set A is a set of (k) subsets of A, denoted as P_A, satisfying the following conditions:

$$(1)\ \emptyset \notin P_A \qquad (2)\ \bigcup_{B \in P_A} B = A \qquad (3)\ \text{if } B, C \in P_A \text{ and } B \neq C \text{ then } B \cap C = \emptyset$$

An ordered partition is a partition in which the subsets have a defined order. For a first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and a domain D, a partition-pair p is a pair of ordered partitions for $\mathcal{R}$ and D, written as $p = \langle (\mathcal{R}_0, D_0), \ldots, (\mathcal{R}_m, D_m) \rangle$, with $k = m+1$ subsets.

Thus, for a given knowledge base, the set of partition-pairs consists of all possible combinations of partitions of the conditionals and the domain. The only restriction is that both partitions need to have the same number of subsets. Among them, only some partition-pairs fulfill the conditions for a tolerance-pair given in Definition 16, which makes the set of tolerance-pairs a subset of the set of partition-pairs. This relation gives rise to a *generate-and-test* algorithm to find all tolerance-pairs, named BruteForceTPair.

In lines 2 to 7 of the algorithm, all possible partition-pairs are created, making up the *generate*-part of the algorithm. The idea behind this approach is that there are certain limits to the number of subsets k a partition-pair can have. Obviously, according to Definition 19, the minimal number of subsets is $k = 1$, with all elements being in just one subset. The other extreme case is a partition in which every element has its own subset. Since we need to partition $\mathcal{R}$ and D into the same number of subsets, the cardinality of the smaller set, $\min(|\mathcal{R}|, |D|)$, defines the maximal number of possible subsets.

Consecutively, the algorithm creates all partitions-pairs from the smallest to the highest possible value for k. For each k, all k-partitions of $\mathcal{R}$ and D are generated, and every combination of them yields a partition-pair. The generation of all ordered k-partitions for a set of n elements, as used in lines 3 and 4 of the algorithm, can be easily done by enumerating the integers from 0 to $k^n - 1$. If these numbers are written in a base-k number system, the ith-digit, which is a number j from 0 to $k - 1$, can be interpreted as the assignment of the ith element of the original set to the jth subset of the partition. Using this approach, every possible k-partition of a set can be created. However, not every number from 0 to $k - 1$ represents a subset-assignment that is a valid partition, as no subset is allowed to be empty. Numbers such as 0, which assign every element of the set to the first subset, leaving the other subsets empty, have to be filtered out.

In lines 8 to 10 of BruteForceTPair, the *test*-part of the algorithm, every partition-pair is checked against the conditions of Definition 16. If a pair does not satisfy them, it is removed from the result set, making sure that only tolerance-pairs are returned by the algorithm. In order to perform this check, algorithm IsTolerancePair from Chapter 4 is used. This approach is demonstrated in the next example.

Example 6 Let a first-order knowledge base be

$$\mathcal{KB}_6 = \langle\ \{\forall x P(x) \Rightarrow B(x)\}\ ,\ \{(F(x)|B(x)), (\bar{F}(x)|P(x))\}\ \rangle$$

and the domain contain the two constants $D = \{p, t\}$. For the sake of legibility, we refer to the conditionals as r_1 and r_2. In this case, partition-pairs have at least 1 and at most $\min(|\mathcal{R}|, |D|) = \min(2, 2) = 2$ subsets. For $k = 1$, the only possible partition-pair is

$$p_1 = \langle\ (\{r_1, r_2\}, \{p, t\})\ \rangle.$$

For $k = 2$, we first look at the way partitions for the sets $\mathcal{R}$ and D are created. The first column shows the integers from 0 to $k^n - 1 = 3$, the second their 2-digit representation

Algorithm 3 BruteForceTPair

Input: $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and D

Output: Set of tolerance-pairs P

```
1: P ← ∅
2: for k ∈ {1, ..., min(|R|, |D|)} do
3:     P_R ← { p_R = ⟨R_0, ..., R_{k-1}⟩ | p_R is an ordered k-partition of R}
4:     P_D ← { p_D = ⟨D_0, ..., D_{k-1}⟩ | p_D is an ordered k-partition of D}
5:     for (p_R, p_D) ∈ P_R × P_D do
6:         p ← partition-pair obtained by combining p_R and p_D
7:         P ← P ∪ {p}
8: for p ∈ P do
9:     if not IsTolerancePair(p, F) then
10:        P ← P \ {p}
```

in base 2 and the third the corresponding subset-assignments for $\mathcal{R}$:

0	00	$\langle \{r_1, r_2\}, \{\} \rangle$
1	01	$\langle \{r_1\}, \{r_2\} \rangle$
2	10	$\langle \{r_2\}, \{r_1\} \rangle$
3	11	$\langle \{\}, \{r_1\} \rangle$

As we can see, only numbers for which every possible value occurs for at least one digit in the base-k representation yield a subset-assignment that is a valid ordered partition. The first and fourth number in this example has no 0 respectively 1 and therefore creates empty subsets. The two remaining assignments are valid ordered partitions. For D, analogously two possible partitions can be obtained. Combining them leads to four partition-pairs:

$$p_2 = \langle\ (\{r_1\}, \{p\})\ ,\ (\{r_2\}, \{t\})\ \rangle$$
$$p_3 = \langle\ (\{r_1\}, \{t\})\ ,\ (\{r_2\}, \{p\})\ \rangle$$
$$p_4 = \langle\ (\{r_2\}, \{p\})\ ,\ (\{r_1\}, \{t\})\ \rangle$$
$$p_5 = \langle\ (\{r_2\}, \{t\})\ ,\ (\{r_1\}, \{p\})\ \rangle$$

The second part of the algorithm tests each of these five partition-pairs against the requirements for a tolerance-pair. In this case, only partition-pairs p_2 and p_3 satisfy the conditions, and are therefore returned by the algorithm. ∎

Clearly, due to the systematic generation of all possible partition-pairs in the algorithm, this approach guarantees to find all tolerance-pairs that exist for a knowledge base. Since every returned pair is explicitly checked against Definition 16, it is also a correct answer.

The algorithm also terminates in every case, as the number of iterations in each loop is given by the sizes of the input sets.

Observation 4 *Algorithm* BruteForceTPair *is sound and complete, that is, for a first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and a domain D, every returned pair is a correct tolerance-pair and every tolerance-pair that exists is returned.*

The main disadvantage of this approach is that every possible partition-pair has to be generated. For each k, every combination of k-partitions of $\mathcal{R}$ and k-partitions of D is a partition-pair, which makes the number of pairs N be

$$N(\mathcal{R}, D) = \sum_{k=1}^{min(|\mathcal{R}|,|D|)} \left(k! \cdot \left\{ {|\mathcal{R}| \atop k} \right\} \cdot k! \cdot \left\{ {|D| \atop k} \right\} \right)$$

with $\left\{ {a \atop b} \right\}$ being the Stirling number of the second kind, that is the number of unordered b-partitions of a set with a elements. As an upper bound, we can use b^a instead due to the way b-partitions are generated by the algorithm. If we define $n = |\mathcal{R}| + |D|$ and hence $min(|\mathcal{R}|, |D|) \leq \frac{1}{2}n$, we can derive an upper bound that only depends on n:

$$N(\mathcal{R}, D) \;\leq\; \sum_{k=1}^{0.5n} k^{|\mathcal{R}|} \cdot k^{|D|} = \sum_{k=1}^{0.5n} k^{|\mathcal{R}|+|D|} = \sum_{k=1}^{0.5n} k^n$$

As Faulhaber's formula tells us, the summation of the nth powers of a sequence of increasing integers is a $n+1$ degree polynomial. Therefore, the number of partition-pairs is in $\mathcal{O}(0.5n^{n+1})$. For the test-part, each pair can be checked in $\mathcal{O}(2^{|\mathcal{P}||D|})$, as we saw in Chapter 4. Combining these partial results, we can see that the asymptotic runtime behavior in terms of n is exponential:

Observation 5 *Algorithm* BruteForceTPair *has exponential time complexity $\mathcal{O}(n^{n+1})$.*

Although this algorithm is too complex to be used on large knowledge bases, its obvious property of completeness makes it useful to study the existence of tolerance-pairs on small examples. In the first-order setting of system Z, there are three components that influence the existence of tolerance-pairs: the set of conditionals $\mathcal{R}$, the size of the domain D and the certain knowledge $\mathcal{F}$. While we can have conflicts between conditionals, as it is the case in system Z, we also need to take care of the dependencies with and between the other components when looking for tolerance-pairs. The following detailed example shows the effects of different D and $\mathcal{F}$ on the existence tolerance-pairs.

Example 7 Let our knowledge base be a simple version of the bird-penguin-scenario that will be gradually extended throughout this example. Initially, in $\mathcal{KB}_7$, we have no facts and only two conditionals, one saying that birds usually fly and one saying that penguins

	$B(p)$	$P(p)$	$F(p)$	$(F(x)\|B(x))$	$(\overline{F}(x)\|P(x))$	$\forall x P(x) \Rightarrow B(x)$
ω_0	0	0	0			
ω_1	0	0	1			
ω_2	0	1	0		+	–
ω_3	0	1	1		–	–
ω_4	1	0	0	–		
ω_5	1	0	1	+		
ω_6	1	1	0	–	+	
ω_7	1	1	1	+	–	

Figure 5.2: Possible worlds for $\mathcal{KB}_7$ and $D = \{p\}$ (+: verified, –: falsified)

usually do not fly:

$$\mathcal{KB}_7 = \langle\ \emptyset\ , \{(F(x)|B(x)), (\overline{F}(x)|P(x))\}\ \rangle$$

If we look at the definition of a tolerance-pair, we can see that there has to be at least one constant in our domain to fulfill condition (2) and (3). If $D = \emptyset$, no tolerance-pair exists. By adding p to the domain, the open conditionals can be instantiated and the conditions can be satisfied. For $\mathcal{KB}_7$ and $D = \{p\}$, one (and only one) tolerance-pair exists:

$$p_1 = \langle\ (\{(F(x)|B(x)), (\overline{F}(x)|P(x))\},\{p\})\ \rangle$$

Interestingly, the two conditionals, which seem to be conflicting because of their contradictory consequences, can be placed into the same subset of the tolerance-pair. This is the case because Definition 16 only requires each conditional to find a world in which it is verified and the other is not falsified, but it does not necessarily have to be the same world. Figure 5.2 shows all possible worlds for the knowledge base. We can see that there is no single world in which both conditionals are verified, yet ω_2 and ω_5 each verify one and do not falsify the other.

Based on the intended meaning of the conditionals, we would expect that only one conditional can apply for an individual, because we know that penguins are a subclass of birds. By adding the rule $\forall x P(x) \Rightarrow B(x)$ to the certain knowledge $\mathcal{F}$, we incorporate this fact into the knowledge base. This leads to $\mathcal{KB}_6$ that was already defined in Example 6. If additional facts are added to the certain knowledge, the effect is that the worlds which can be used to verify a conditional are reduced, as only worlds satisfying all formulas in $\mathcal{F}$ are considered. In our example, the new formula is not valid in worlds ω_2 and ω_3 as depicted in the last column of Figure 5.2. That means we no longer have a world to fulfill the tolerance-pair conditions for the second conditional and hence, for $\mathcal{KB}_6$ and $D = \{p\}$, no tolerance-pair exists. Due to the added certain knowledge, the conditionals are now indeed conflicting conditionals and have to be placed in separate subsets.

The conflict between $(F(x)|B(x))$ and $(\overline{F}(x)|P(x))$ can be resolved by looking at a greater universe, in which both, a flying bird and a non-flying penguin, can exist. For a domain $D = \{p, t\}$, two tolerance-pairs exist, as we already saw in Example 6:

$$p_2 = \langle\ (\{(F(x)|B(x))\},\{p\}),\\ (\{(\overline{F}(x)|P(x))\},\{t\})\ \rangle$$

$$p_3 = \langle\ (\{(F(x)|B(x))\},\{t\}),\\ (\{(\overline{F}(x)|P(x))\},\{p\})\ \rangle$$

The conflicting conditionals are now separated into subsets, and, due to the fact that all penguins are birds, also separated according to their specificity. The general conditional about birds is in the first subset, whereas the more specific penguin-conditional is in the second. Analogously to system Z, this in ensured by condition (3) of Definition 16, requiring that conditionals in higher-labeled subsets are not falsified in the world in which a conditional is verified. The constants, however, can be arbitrarily placed into the subsets, as we do not have any specific knowledge about them. If we also add facts about p and t, we further restrict the possible worlds that are taken into account to fulfill the tolerance-pair conditions. With p being a bird and t being a penguin, as in

$$\mathcal{KB}_8 = \langle\ \{\forall x P(x) \Rightarrow B(x), B(p), P(t)\}\ , \{(F(x)|B(x)), (\overline{F}(x)|P(x))\}\ \rangle,$$

we have only one possible tolerance-pair for $D = \{p, t\}$ left, which is p_2.

While adding more facts to a knowledge base seems to reduce the number of tolerance-pairs, adding additional constants to the domain without any certain knowledge can significantly increase the number of tolerance-pairs. For $\mathcal{KB}_8$ and $D = \{p, t, a\}$, for instance, three tolerance-pairs exist, as opposed to only one without a:

$$p_4 = \langle\ (\{(F(x)|B(x))\},\{p, a\}),\\ (\{(\overline{F}(x)|P(x))\},\{t\})\ \rangle$$

$$p_5 = \langle\ (\{(F(x)|B(x))\},\{a\}),\\ (\{(\overline{F}(x)|P(x))\},\{p, t\})\ \rangle$$

$$p_6 = \langle\ (\{(F(x)|B(x))\},\{p\}),\\ (\{(\overline{F}(x)|P(x))\},\{t, a\})\ \rangle$$

For larger knowledge bases, this effect can lead to an even higher number of new tolerance-pairs. The new constant a can be placed in each subset, and, due to the lack of any information on whether it is a bird or a penguin, it can serve to find a verifying world for each conditional. In p_5, the bird-conditional is verified with a and the bird, p, is placed into the second subset together with the penguin-conditional. This seems to be an

undesirable result, as the specificity among the constants p and t, which is clearly given by the certain knowledge, is not represented in the partition. Though, for p_5, the conditions of Definition 16 are satisfied in the following worlds

$$\omega_0 = B(p)B(t)B(a)\overline{P}(p)P(t)\overline{P}(a)\overline{F}(p)\overline{F}(t)F(a)$$
with $\omega_0 \models \mathcal{F}$, ω_0 verifies $(F(a)|B(a))$ and ω_0 does not falsify $(\overline{F}(a)|P(a))$
$$\omega_1 = B(p)B(t)\overline{B}(a)\overline{P}(p)P(t)\overline{P}(a)\overline{F}(p)\overline{F}(t)\overline{F}(a)$$
with $\omega_1 \models \mathcal{F}$, ω_1 verifies $(\overline{F}(t)|P(t))$ and ω_1 does not falsify $(\overline{F}(p)|P(p))$

and it is therefore a valid tolerance-pair. The effect of new, unspecified constants on the number of tolerance-pairs grows with the size of the knowledge base. For $\mathcal{KB}_4$ and $D = \{p, t, s\}$ we found 3 tolerance-pairs in Example 4. Adding one unspecified constant a to D increases the number of tolerance-pairs to 43. ∎

This detailed example showed the different effects of certain knowledge and the size of the domain on the existence of tolerance-pairs. Although the findings in this section are more informal, we still want to summarize them in an observation. A knowledge engineer modeling a first-order knowledge base for system Z^{FOL} can use it as a guidance in order to obtain tolerance-pairs that match the intended meaning.

Observation 6 *For a first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and a domain D, ...*

1. *... none, one or multiple tolerance-pairs can exist.*
2. *... the less tolerance-pairs exist, the more certain knowledge $\mathcal{F}$ is added.*
3. *... the domain has to be large enough to provide an individual for each conflicting conditional, to ensure that a tolerance-pair exists.*
4. *... adding unspecified constants increases number of tolerance-pairs significantly.*

Part (2) of the observation takes note of the fact that additional certain knowledge reduces the number of possible worlds that are considered to justify the placement of a conditional in a certain subset. Regarding the size of the domain, (3) and (4) capture the two extremes that were illustrated by the example, leading to either none or too many tolerance-pairs. The reason of (3) is basically that every open conditional needs a representative, and, for conflicting conditionals, they have to be different individuals. The notion of *conflicting conditionals* is however much more complex than in the propositional case. While the last example already pointed out that a conflict is not necessarily given by the conditionals alone but in combination with the facts, it is actually even more intricate. A conflict may arise between conditionals that make statements about classes of individuals, as it is the case in the example, but also between conditionals about certain individuals or an individual and its class. In order to find such conflicts, also the domain is relevant.

To handle the *unspecified-constant-problem* expressed by (4), we could avoid to add such constants to the domain. Although this seems to be a practical way to cope with this issue, it is contrary to the idea of default reasoning, as Lehman and Magidor point out:

> "Default reasoning (in the largest possible meaning) is mainly useful when the number of individuals is very large, i.e., when we do not expect to have an exhaustive list of all those individuals available. Otherwise, it seems we could get full information and then classical monotonic reasoning is called for." [LM90, p. 63]

As this argument does not fully apply to system Z^{FOL}, which requires us to always provide a finite list of all individuals, the effect of unspecified constants is definitely an issue that has to be addressed.

Finally, a last example illustrates another aspect of the existence of tolerance-pairs. In the previous example, in several cases the number of tolerance-pairs could be reduced by adding additional facts or adjusting the domain in order to obtain only the desired tolerance-pair(s). In the next knowledge base though, this is different.

Example 8 Let $\mathcal{KB}_9$ be the knowledge base $\mathcal{KB}_8$ with one additional conditional saying that birds usually have wings:

$$\mathcal{KB}_9 = \langle\ \{\forall x P(x) \Rightarrow B(x), B(p), P(t)\}\ , \{(F(x)|B(x)), (\overline{F}(x)|P(x)), (W(x)|(B(x))\}\ \rangle$$

For this knowledge base and $D = \{p, t\}$, two tolerance-pairs exist:

$$\begin{aligned} p_7 = \langle\ &(\{(F(x)|B(x)), (W(x)|B(x))\}, \{p\}), \\ &(\{(\overline{F}(x)|P(x))\}, \{t\})\ \rangle \end{aligned}$$

$$\begin{aligned} p_8 = \langle\ &(\{(F(x)|B(x))\}, \{p\}), \\ &(\{(\overline{F}(x)|P(x)), (W(x)|B(x))\}, \{t\})\ \rangle \end{aligned}$$

Although we have exactly two individuals to resolve the flying-conflict and corresponding facts, due to the third conditional, we still have two different tolerance-pairs. However, there seems to be no other information that we could add about having wings, since it should apply for both birds and penguins. ∎

Despite both tolerance-pairs being valid, there is a preferred one among the two. Since the new conditional is a general one about birds, it should be placed into the first subset together with the other bird-conditional and the bird-individual. Hence, the notion of tolerance-pairs alone might not be sufficient, and instead, we need to look at a more restricted subset. In the propositional case, similarly, only the unique partition that leads to the minimal ranking function is considered.

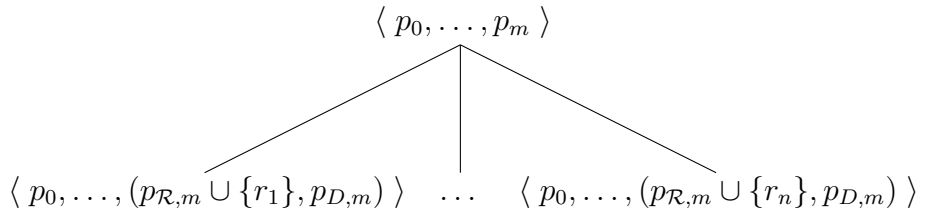
5.3 Finding tolerance-pairs

In this section, we present two algorithms that can be used to find tolerance-pairs for a first-order knowledge base and that are implemented in the system demonstrated in the previous chapter. The first one, algorithm SearchTPair, is an alternative to the brute-force algorithm introduced before. The second, algorithm SearchMinimalTPair, is a variant of it that finds only a specific subset of the tolerance-pairs for a knowledge base.

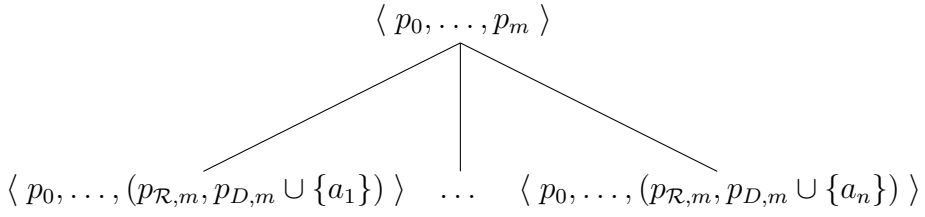
In contrast to the generate-and-test approach of algorithm BruteForceTPair, algorithm SearchTPair is a *backtracking-search* algorithm that explores a search tree obtained by incrementally constructing tolerance-pairs. Every node in the search tree represents a partition-pair for a subset of the conditionals and a subset of the domain. Its children are partition-pairs created by adding additional conditionals or constants to it. Once all conditionals and constants have been added, a partition-pair of the knowledge base results and we reach a leaf of the search tree. In this tree, the algorithm performs a depth-first search to find tolerance-pairs. It backtracks if it finds a partition-pair for which it is no longer possible to yield a valid tolerance-pair.

If we look at an arbitrary partition-pair $p = \langle p_0, \ldots, p_m \rangle$, which partitions the subsets $\mathcal{R}' \subseteq \mathcal{R}$ and $D' \subseteq D$, there are several ways to extend it to new partition-pairs. As the pairs are constructed incrementally, starting with $\langle (\emptyset, \emptyset) \rangle$, we only add new elements to the highest tuple p_m in each step. The following extensions are possible:

1. *Add a conditional:* For each conditional $r_i \in (\mathcal{R} \setminus \mathcal{R}')$, with $1 \leq i \leq n, n = |\mathcal{R} \setminus \mathcal{R}'|$, a new partition-pair $p' = \langle p_0, \ldots, (p_{\mathcal{R},m} \cup \{r_i\}, p_{D,m}) \rangle$ can be created:



2. *Add a constant:* For each constant $a_i \in (D \setminus D')$, with $1 \leq i \leq n, n = |D \setminus D'|$, a new partition-pair $p' = \langle p_0, \ldots, (p_{\mathcal{R},m}, p_{D,m} \cup \{a_i\}) \rangle$ can be created:

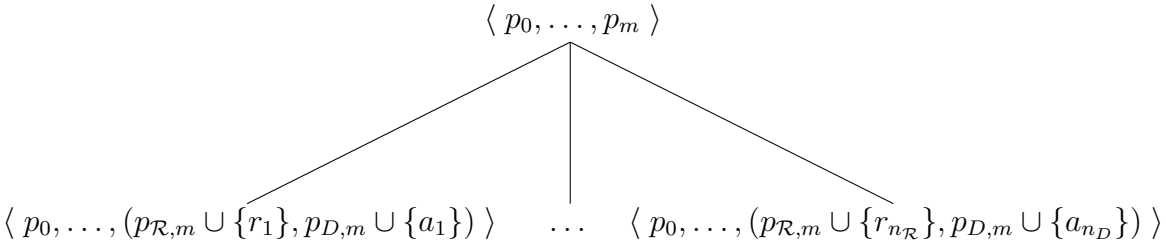


3. *Enlarge the partition-pair to $m+1$:* A new tuple $p_{m+1} = (\emptyset, \emptyset)$ can be added, yielding $p' = \langle p_0, \ldots, p_m, (\emptyset, \emptyset)\rangle$:

$$\begin{array}{c} \langle\ p_0, \ldots, p_m\ \rangle \\ | \\ \langle\ p_0, \ldots, p_m, (\emptyset, \emptyset)\ \rangle \end{array}$$

   As we need to have every set $p_{\mathcal{R},i}$ and $p_{D,i}$ for $0 \leq i \leq m$ filled in a partition-pair, this type of extension is only applicable if $p_{\mathcal{R},m}$ and $p_{D,m}$ both already contain at least one element. This makes sure that only the highest-labeled tuple, p_{m+1} after this extension, can temporarily have empty sets. In addition, it does not make sense to enlarge a partition-pair if there are not at least one conditional and one constant left that can be placed in the new tuple p_{m+1}.

4. *Add a conditional and a constant:* For each $(r_i, a_j) \in (\mathcal{R} \setminus \mathcal{R}') \times (D \setminus D')$, with $n_{\mathcal{R}} = |\mathcal{R} \setminus \mathcal{R}'|$ and $n_D = |D \setminus D'|$, a new partition-pair $p' = \langle p_0, \ldots, (p_{\mathcal{R},m} \cup \{r_i\}, p_{D,m} \cup \{a_j\})\rangle$ can be created:



   Although we could achieve to add a conditional and a constant by subsequently performing extensions (1) and (2), there are situations in which it makes sense to immediately add a conditional and a constant. If both $p_{\mathcal{R},m}$ and $p_{D,m}$ are empty, which is the case for the root node of the search tree and if a new tuple has been added by (3), adding both ensures we have a valid partition-pair with no empty set and that the conditions for a tolerance-pair can be checked for it.

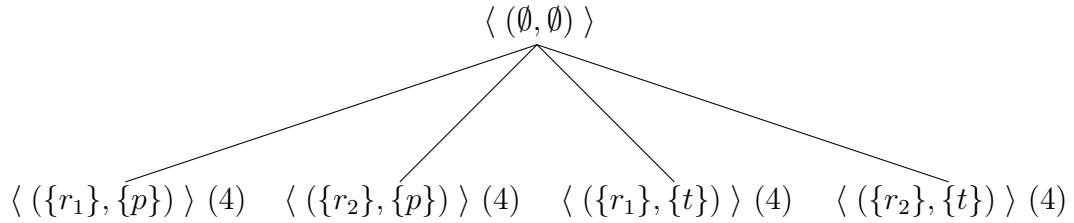
Based on the restrictions mentioned for extension (3) and (4), the construction of the search tree can now be fully described. For each node, either (4) or both (1) and (2), and in addition, possibly (3), can be applied. Starting with the partition-pair $\langle(\emptyset, \emptyset)\rangle$ and recursively applying this strategy, a search tree evolves. A path ends if no further extension is possible because all conditionals and all constants are already added. The leaf node then is a partition-pair for the knowledge base.

Example 9 We look at the same knowledge base as we did for the brute-force algorithm. Let the domain be $D = \{p, t\}$ and the knowledge base be

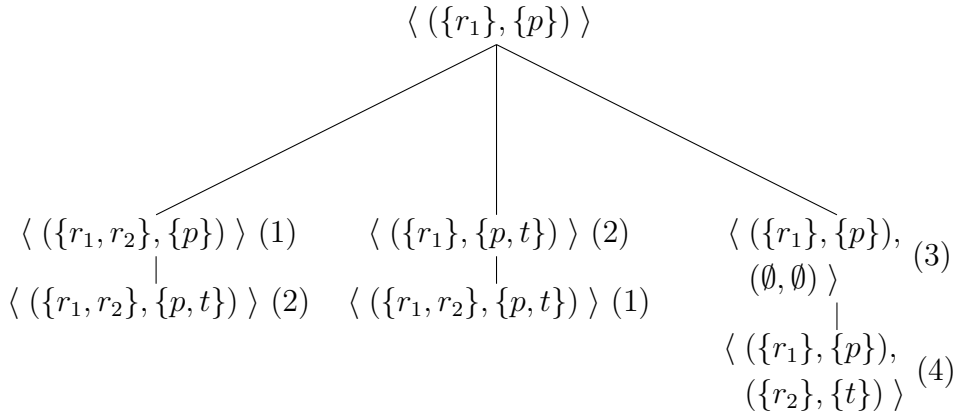
$$\mathcal{KB}_6 = \langle\ \{\forall x P(x) \Rightarrow B(x)\}\ ,\ \{(F(x)|B(x)), (\bar{F}(x)|P(x))\}\ \rangle$$

Again, we refer to the conditionals as r_1 and r_2. Since the root node of the search tree is

empty, only extension (4) can be applied, producing four new nodes:



If we continue with the first child node, we can apply (1) and (2) as well as (3). For each of the created nodes, a last extension can be applied, producing three leaf nodes:



Analogously, the subtrees of the other three child nodes of the root can be created. While the first two leaf nodes are the same among all subtrees, the third one is unique in each. All in all, this results in five distinct partition-pairs. ∎

Observation 7 *Let $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a first-order knowledge base and D be the domain. Every leaf node of the search tree built from $\langle (\emptyset, \emptyset) \rangle$ by extensions (1), (2), (3) and (4) according to the described strategy represents a partition-pair of $\mathcal{KB}$ and D and every possible partition-pair of $\mathcal{KB}$ and D is represented by a leaf node in this tree.*

Proof According to Definition 19, a partition-pair is a pair of ordered partitions for $\mathcal{R}$ and D. In the search tree, a leaf node is reached if every element of $\mathcal{R}$ and D has been added. Since extension (1), (2) and (4) reduce the amount of either conditionals or constants that are left to add, and because extension (3) can only be used with one of the others before and after, every path in the tree is finite.

At a leaf node, (2) of Definition 19 holds, and, since every element is added only once, also (3). Empty sets temporarily occur if a new tuple is added by extension (3), but, as the restrictions for this extension requires that there have to be enough conditionals and constants left to fill the new tuple and that it can only be added if the current tuple is filled, at a leaf node, every subset of the partitions of $\mathcal{R}$ and D is filled. Hence, also (1) of Definition 19 holds, that means, we create proper partitions of $\mathcal{R}$ and D. Due

to their incremental construction the partitions are obviously ordered. Therefore, a leaf node represents a valid partition-pair for $\mathcal{KB}$ and D as defined in Definition 19.

For every node of the search tree, each of the possible extensions is applied, defining a new path of the tree. The restrictions regarding the application of extensions (3) and (4) ensure that only proper partition-pairs are created, as argued above, but do not further restrict the construction of possible partition-pairs. Therefore, the full search tree represents every possible combination of adding conditionals and constants to the partition-pair and its leaf nodes correspond to every partition-pair that exists for the knowledge base. ∎

By now, the new search tree approach just provides another way to create all partition-pairs of a knowledge base. However, the advantage of this approach is that the tree search can be enhanced with a logic that checks for every node whether or not the partial solution constructed so far can be extended to a tolerance-pair for the knowledge base. If not, the complete subtree of this node can be skipped. Hence, in contrast to algorithm BruteForceTPair, this algorithm does not necessarily generate every partition-pair to find all tolerance-pairs. As it is shown in the pseudo code, in function SEARCH the check is done by calling TEST for each node in line 4. Only if TEST returns a positive result, the node's partition-pair is extended according to the search strategy in lines 6 to 19 and the new nodes are processed recursively. Otherwise, this node becomes a leaf node of the tree and the algorithm backtracks to the next node.

In order to decide whether a node should be expanded, function TEST performs different checks. First, the special case of an empty tuple $p_m = (\emptyset, \emptyset)$, that occurs at the tree's root node and after extension (3), is handled in lines 21 and 22. In this case, the partition-pair should definitely be further extended, and no other checks need to be performed. The second check is included to cope with a problem that already occurred in Example 9: For the second child node of the root, one possible extension is to add r_1, which leads to the same partition-pair as adding r_2 to the first child node of the root. In general, there are paths that result in the same partition-pair and just differ in the order the elements have been added to the partition-pairs. Therefore, the algorithm has a data structure V that stores every partition-pair constructed so far. In lines 23 to 26, it is checked whether a partition-pair has already been created before. If this is the case, the node does not have to be expanded again, since the subtree is equal to a subtree already explored. The third check in lines 27 and 28 tests whether the partition-pair can be extended to a tolerance-pair for the knowledge base. If this is not the case, the node is not expanded. If it is, and in addition, all conditionals and constants are already part of the partition-pair, it is in fact a tolerance-pair and is added to the result set. Otherwise, the node is further expanded. The algorithm stops if there is no node left in the tree that can be expanded.

For the third check, a modified version of the tolerance-pair check implemented in algorithm IsTolerancePair is used. The idea is that, in order to check whether the conditions of Definition 16 hold for a conditional, we only have to know the subset $p_{\mathcal{R},i}$ of $\mathcal{R}$ it is placed in, the corresponding constants $p_{D,i}$ and the conditionals that are in

Algorithm 4 SearchTPair

Input: $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and D
Output: Set of tolerance-pairs P

$P \leftarrow \emptyset, V \leftarrow \emptyset$
search$(\langle(\emptyset, \emptyset)\rangle, 0, \mathcal{R}, D)$
function search$(p, i, \mathcal{R}_L, D_L)$
  if not test$(p, i, \mathcal{R}_L, D_L)$ **then**
    return
  if $p_{\mathcal{R},i} = \emptyset$ **and** $p_{D,i} = \emptyset$ **then** ▷ extension (4)
    for $(r, a) \in \mathcal{R}_L \times D_L$ **do**
      $p' \leftarrow p$ with $p'_{\mathcal{R},i} \leftarrow p_{\mathcal{R},i} \cup \{r\}$ and $p'_{D,i} \leftarrow p_{D,i} \cup \{a\}$
      search$(p', i, \mathcal{R}_L \setminus \{r\}, D_L \setminus \{a\})$
  else
    for $r \in \mathcal{R}_L$ **do** ▷ extension (1)
      $p' \leftarrow p$ with $p'_{\mathcal{R},i} \leftarrow p_{\mathcal{R},i} \cup \{r\}$
      search$(p', i, \mathcal{R}_L \setminus \{r\}, D_L)$
    for $a \in D_L$ **do** ▷ extension (2)
      $p' \leftarrow p$ with $p'_{D,i} \leftarrow p_{D,i} \cup \{a\}$
      search$(p', i, \mathcal{R}_L, D_L \setminus \{a\})$
  if $p_{\mathcal{R},i} \neq \emptyset$ **and** $p_{D,i} \neq \emptyset$ **and** $\mathcal{R}_L \neq \emptyset$ **and** $D_L \neq \emptyset$ **then** ▷ extension (3)
    $p' \leftarrow \langle p_0, \ldots, p_i, (\emptyset, \emptyset)\rangle$
    search$(p', i + 1, \mathcal{R}_L, \mathcal{D}_L)$
function test$(p, i, \mathcal{R}_L, D_L)$
  if $p_{\mathcal{R},i} = \emptyset$ **and** $p_{D,i} = \emptyset$ **then** ▷ no tests for empty tuple
    return `true`
  if $p \in V$ **then** ▷ node already expanded?
    return `false`
  else
    $V \leftarrow V \cup \{p\}$
  if not IsPotentialTPair$(p, \mathcal{F}, \mathcal{R}_L)$ **then** ▷ can be a tolerance-pair?
    return `false`
  if $\mathcal{R}_L = \emptyset$ **and** $D_L = \emptyset$ **then** ▷ tolerance-pair found
    $P \leftarrow P \cup \{p\}$
    return `false`
  return `true` ▷ expand this node

Algorithm 5 IsPotentialTPair

Input: potential tolerance-pair p, facts $\mathcal{F}$, left conditionals $\mathcal{R}_L$

Output: `true`, if conditions (1), (2) and (3) of Definition 16 hold for every $r \in p_{\mathcal{R},m}$, otherwise, `false`

```
1: for r ∈ p_{R,m} do
2:     hasWorld ← false
3:     for ω ∈ Ω with ω ⊨ F do
4:         for a ∈ p_{D,m} do
5:             if ω verifies r(a) and not fals. r'(a') for all r' ∈ p_{R,m} ∪ R_L, a' ∈ p_{D,m} then
6:                 hasWorld ← true
7:     if not hasWorld then
8:         return false
9: return true
```

higher subsets. To check condition (3) for a conditional that has just been placed in a partition-pair that is constructed, we can assume that all conditionals not yet added will be placed in the same or a higher-labeled subset, but we do not need the information in which exact subset they will be placed. Therefore, although the partition-pair is not yet fully built, we can already check the conditions for the added conditional. This is the idea of algorithm IsPotentialTPair, in which the differences to IsTolerancePair are underlined. We can see that only the conditionals of the highest-labeled subset are checked in the outer loop in line 1, ignoring all conditionals in lower subsets. In order to test for falsifications in line 5, only these conditionals and the conditionals that are left to be added are considered. The following observation shows that the algorithm can be used in SearchTPair to ensure that every leaf node is a tolerance-pair:

Observation 8 *Let $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a first-order knowledge base, D be the domain and $p = \langle p_0, \ldots, p_m \rangle$ be a partition-pair for $\mathcal{KB}$ and D that has been incrementally created in the way algorithm* SearchTPair *explores the search tree. Partition-pair p is a tolerance-pair for $\mathcal{KB}$ and D if* IsPotentialTPair *returns a positive result, evaluated as*

$$\text{IsPotentialTPair}(p, \mathcal{F}, \mathcal{R} \setminus \cup_{i=0}^{m} p_{\mathcal{R},i}),$$

each time a conditional or constant has been added to p.

Proof In a leaf node of the search tree explored by algorithm SearchTPair, all conditionals and constants are added to the partition-pair, that is, $\cup_{i=0}^{m} p_{\mathcal{R},i} = \mathcal{R}$ and $\cup_{i=0}^{m} p_{D,i} = D$. During the construction, they are split into two sets, the ones already in the partition-pair, and those which are still left. The check for a fully-built partition-pair

requires a world in which a conditional $r \in p_{\mathcal{R},i}$ is verified for a constant from $p_{D,i}$ and no conditionals from any $p_{\mathcal{R},j}, j \geq i$, for constants from $p_{D,i}$, is falsified.

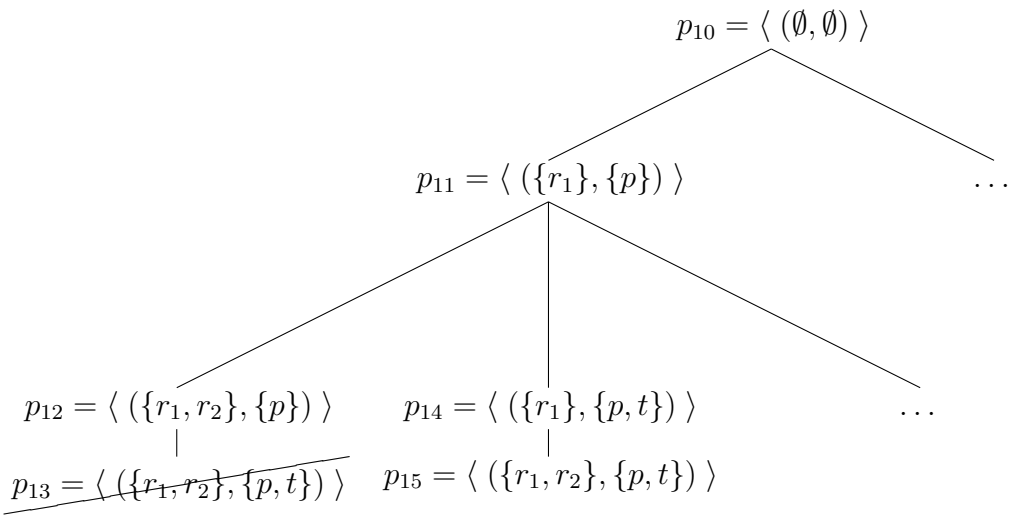
When adding a conditional to a partially-built partition-pair, algorithm IsPotentialTPair checks the conditionals already added to the current subset $p_{\mathcal{R},m}$ and the left conditionals $\mathcal{R} \setminus \cup_{i=0}^{m}$. As conditionals and constants are incrementally added to the tuples p_i from $i = 0$ to $i = m$, only the left conditionals can be added to $p_{\mathcal{R},j}, j \geq i$. Therefore, the check in algorithm IsPotentialTPair takes the same conditionals into account as a check for the fully built partition-pair by algorithm IsTolerancePair. For each last check for an $i \in \{0, \ldots, m\}$, executed before a new tuple is added or after the partition-pair is fully built, both $p_{\mathcal{R},i}$ and also $p_{D,i}$ are final, and for each $r \in p_{\mathcal{R},i}$ algorithm IsPotentialTPair does exactly the same check as algorithm IsTolerancePair. Hence, a fully-built partition-pair that passed every check is a tolerance-pair as defined in Definition 16. ∎

We just showed that using IsPotentialTPair, we can find paths in the search tree that lead to proper tolerance-pairs. An open question is whether the opposite is true, that is, if the algorithm fails for a node in the search tree, that there is no tolerance-pair in the subtree. The next example shows that this is not the case:

Example 10 Let our knowledge base be

$$\mathcal{KB}_{10} = \langle\ \{\overline{B}(t), \overline{P}(p)\}\ ,\ \{(F(x)|B(x)), (\bar{F}(x)|P(x))\}\ \rangle$$

and the domain be $D = \{p, t\}$. In this case, the facts say individual p is not a penguin and individual t is not a bird, while we not say that a penguin is also a bird. We refer to the conditionals as r_1 and r_2. A depth-first search in the corresponding search tree first applies extension (4) and processes the first child node p_{11}. Here, extension (1) can be applied, adding the second conditional r_2 and receiving p_{12}:



At this point, algorithm IsPotentialTPair returns a negative result for p_{12}, as there is no world in which $r_2(p)$ is verified, which is directly in conflict with the facts. The search does not explore the child node p_{13} but backtracks to the parent node p_{11} and processes the next children. However, the leaf node p_{13} in the first path which is never processed is indeed a tolerance-pair: In world

$$\omega_1 = B(p)\overline{B}(t)\overline{P}(p)\overline{P}(t)F(p)\overline{F}(t),$$

the facts hold and conditional $r_1(p)$ is verified and neither $r_2(p)$ nor $r_1(t)$ nor $r_2(t)$ is falsified. In addition, in world

$$\omega_2 = \overline{B}(p)\overline{B}(t)\overline{P}(p)P(t)\overline{F}(p)\overline{F}(t),$$

the facts hold as well and conditional $r_2(t)$ is verified and none of $r_1(t)$, $r_1(p)$ and $r_2(p)$ is falsified. That means, although the path would result in a tolerance-pair, IsPotentialTPair fails for p_{12}. The reason is that the constant t which in necessary for conditional r_2 to be successfully checked is added too late. ∎

Based on the example, we might conclude that using algorithm IsPotentialTPair to prune the search tree can make the search algorithm incomplete, as paths leading to tolerance-pair could be cut. But if we look at the reason for the behavior in the example, we can easily see that this is not a problem: If the constant t had been added before conditional r_2, as it is shown in the path of partition-pair p_{14} and p_{15}, the checks would have been successful. And as the search tree generates a path for each possible order in which conditionals and constants can be added, there is always a path in which the tolerance-pair is found, even if the subtrees are skipped due to IsPotentialTPair failing. These results are now summarized:

Observation 9 *Algorithm* SearchTPair *is sound and complete, that is, for a first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and a domain D, every returned pair is a correct tolerance-pair and every tolerance-pair that exists is returned.*

Proof In Observation 7, we showed that the leaf nodes of the full search tree correspond to the possible partition-pairs of the knowledge base. During the proof we also argued that the tree is finite and a depth-first search in it therefore always terminates. In algorithm SearchTPair, the full search tree is pruned in different cases by function TEST returning `false`. In order to prove this observation, we now have to show that each leaf node of the original tree that is still reached is in fact a tolerance-pair and that none of them is missed due to the pruning.

In lines 23 to 26, nodes which have already been processed are skipped. Since every node is processed in the same way, the skipped subtree is completely equal to a subtree that has already been processed, hence, this does not influence soundness and completeness of the search. In Observation 8, we showed that pruning by algorithm IsPotentialTPair

in lines 27 and 28 makes sure that only correct tolerance-pairs are found and after the observation we argued that this does not influence the completeness. The last case in which function TEST returns a negative result in line 31 is trivial, as this is a leaf node of the full search tree anyway. Therefore, algorithm SEARCHTPAIR finds every tolerance-pair for $\mathcal{KB}$ and D. ∎

The algorithm's time complexity depends on the size of the search tree. If we look at extensions (1) and (2), we can see that for a problem of size $n = |\mathcal{R}_L| + |D_L|$, they create n subproblems of size $n - 1$. A tree built only by these two extensions would have $n!$ leaf nodes. While extension (4) reduces this number, extension (3) introduces additional subtrees. Although no detailed analysis of the complexity should be given here, we can see that the search tree must be at least as big as the number of partition-pairs according to Observation 7. Therefore, similar to the brute-force approach, a full exploration of the tree has an exponential runtime complexity, as it is typical of backtracking algorithms.

Observation 10 *Algorithm* SEARCHTPAIR *has exponential time complexity.*

The advantage of this algorithm comes with the different pruning strategies incorporated into the depth-first search. An empirical comparison that illustrates the pruning effects is shown in an example in the next section. Next, as it was already mentioned in Example 8, we look at tolerance-pairs of a certain kind and present an algorithm to find only them.

5.4 Minimal tolerance-pairs

In system Z, CONSISTENCYCHECK finds a partition of $\mathcal{R}$ in which every $r \in \mathcal{R}_i$ is tolerated by the set $\cup_{j=i}^{m} \mathcal{R}_j$. In general, several partitions can exist that have this property. But among them, the algorithm finds a unique one in which every subset of the partition, from 0 to m, contains as many conditionals as possible. This minimal partition is then used to compute the ranking function κ_z. For system Z^{FOL}, we introduce a similar notion of *minimal tolerance-pairs*. Analogously, from tuple p_0 to p_m, as many conditionals and constants as possible should be placed in the subsets.

Definition 20 (Minimal Tolerance-Pair)
For a first-order knowledge base $\mathcal{KB}$ and a domain D, let P denote the set of possible tolerance-pairs. A binary relation $<_{TP}$ on $P \times P$ is defined for $p_1, p_2 \in P$ as

$p_1 <_{TP} p_2$ *if and only if*
(1) $m_1 < m_2$ *or*
(2) $m_1 = m_2 = m$ *and there is* $0 \leq i \leq m$:
$|(p_1)_{\mathcal{R},i}| > |(p_2)_{\mathcal{R},i}|$ *or,* $|(p_1)_{\mathcal{R},i}| = |(p_2)_{\mathcal{R},i}|$ *and* $|(p_1)_{D,i}| > |(p_2)_{D,i}|$,

$$\textit{and for all } 0 \leq j < i : |(p_1)_{\mathcal{R},j}| = |(p_2)_{\mathcal{R},j}| \textit{ and } |(p_1)_{D,j}| = |(p_2)_{D,j}|,$$

yielding a strict total order over the set of tolerance-pairs, stating that p_1 *is smaller than* p_2*. A corresponding notion of equality is defined as* $p_1 =_{TP} p_2$ *iff* $p_1 \not<_{TP} p_2$ *and* $p_2 \not<_{TP} p_1$*. A tolerance-pair* p *is minimal for* $\mathcal{KB}$ *and* D *iff there is no* $p' \in P$ *with* $p' <_{TP} p$*.*

The rationale behind choosing the minimal partition in system Z to compute the ranking function is the objective of *highest plausibility*. That means, unless we have explicit information of the contrary, we assume every world to be as plausible as possible. This is achieved by using the minimal partition, since the unique ranking function κ_z based on it has the lowest ranks of all models of the knowledge base, as Theorem 2 states. If we would choose another partition, the ranks would be higher for some worlds, making them less plausible, although this is not required by the knowledge base.

In contrast to system Z, there is not a unique minimal tolerance-pair in system Z^{FOL} for every knowledge base. The following example illustrates this phenomenon:

Example 11 We look at some of the tolerance-pairs shown in Example 7 and Example 8 and identify the minimal ones. For $\mathcal{KB}_9$ and $D = \{p, t\}$, we found two possible tolerance-pairs in Example 8. The reason is that the wings-conditional can be placed in each of the two subsets. We argued that we prefer the one having the conditional in the first subset, as it is a general one about birds, and this one is also preferred by our defined order, as $p_7 <_{TP} p_8$ holds. In Example 7, for $\mathcal{KB}_8$ and $D = \{p, t, a\}$, we found three tolerance-pairs. Here, we also have a unique minimal tolerance-pair p_4, since $p_4 <_{TP} p_5$ and $p_4 <_{TP} p_6$. As a last example, for $\mathcal{KB}_6$ and $D = \{p, t\}$, we found two tolerance-pairs p_2 and p_3. However, none of them is smaller than the other, $p_2 =_{TP} p_3$, so both are minimal in this case. ∎

The example showed that there are situations in which we have several minimal tolerance-pairs, and among them, none seems to be preferable. Therefore, we cannot guarantee to find a unique minimal tolerance-pair in system Z^{FOL} based on Definition 20. A next question is whether the introduced notion of minimality assures that for a minimal tolerance-pair, also the resulting ranking function has most plausible ranks. As a first result, a relation between the order of tolerance-pairs and the ranks of the corresponding ranking functions can be established, as shown in the following observation:

Observation 11 *Let* P *be the set of tolerance-pairs for a first-order knowledge base* $\mathcal{KB}$ *and a domain* D*. It seems to hold that, if* $p_1 <_{TP} p_2$ *for* $p_1, p_2 \in P$*, then* $\kappa_{z_1}^{FOL}(\omega) \leq \kappa_{z_2}^{FOL}(\omega)$ *for each* $\omega \in \Omega$*.*

Although this connection was found to hold for every analyzed knowledge base, a formal proof of it has to be left for further work. As a consequence of the observation, the ranking functions induced by minimal tolerance-pairs in fact assign the most plausible

	$B(p)$	$B(t)$	$P(p)$	$P(t)$	$F(p)$	$F(t)$	$\kappa_z^{FOL}(\omega)$	$\kappa'(\omega)$	$\kappa''(\omega)$
ω_0	1	1	0	1	0	0	1	1	2
ω_1	1	1	0	1	0	1	4	2	0
ω_2	1	1	0	1	1	0	0	0	2
ω_3	1	1	0	1	1	1	3	2	0
ω_4	1	1	1	1	0	0	1	1	2
ω_5	1	1	1	1	0	1	4	2	0
ω_6	1	1	1	1	1	0	2	1	2
ω_7	1	1	1	1	1	1	5	2	1

Figure 5.3: Ranking functions of Example 12 for $\kappa(\omega) \neq \infty$

ranks to worlds, compared to those computed for other tolerance-pairs. If several minimal tolerance-pairs exists, there are usually some worlds ranked lower by one of them and other worlds ranked the other way round, but none of them is – overall – more plausible than the other. While minimal tolerance-pairs seem to be a suitable way to choose among the set of tolerance-pairs, they do not lead to a most plausible model as in the propositional case. As the next example shows, the minimal model of a knowledge base might not be induced by a tolerance-pair at all:

Example 12 Again, we look at

$$\mathcal{KB}_8 = \langle \{\forall x P(x) \Rightarrow B(x), B(p), P(t)\} , \{(F(x)|B(x)), (\overline{F}(x)|P(x))\} \rangle$$

and $D = \{p, t\}$ from Example 7, for which one tolerance-pair exists:

$$p_2 = \langle (\{(F(x)|B(x))\}, \{p\}) , (\{(\overline{F}(x)|P(x))\}, \{t\}) \rangle$$

In Figure 5.3, the ranking function κ_z^{FOL} that can be computed for the tolerance-pair is shown. It accepts both conditionals, with p being a representative for the bird-conditional and t for the penguin-conditional and is the only model provided by system Z^{FOL}, as only one (minimal) tolerance-pair exists in this case

Both κ' and κ'' in Figure 5.3 are also models of the knowledge base. For κ', p is a representative for the bird-conditional, which is accepted fulfilling (Acc-2) from Definition 12. It is falsified for p more plausibly (rank 1) that its negation is falsified for its representative t (rank 2). The penguin-conditional has the representative t and is accepted with (Acc-1), as its verification is ranked more plausibly (rank 0) than its falsification (rank 1). For each world, the rank assigned by κ' is less then or equal to κ_z^{FOL}.

For κ'', the bird-conditional is accepted by (Acc-2) due to the lack of having a representative for its negation. Its representative is t. The penguin-conditional has p as a

representative and t for its negation, being falsified with the ranks 1 and 2, and therefore fulfilling (Acc-2). Compared to κ', the representatives are swapped in this model. ∎

As the example pointed out, system Z^{FOL} does not necessarily generate a minimal model for a knowledge base, as it is the case for system Z. The assigned ranks could still be lowered while maintaining the property of being a model. Hence, the notion of a minimal tolerance-pair, and even of a tolerance-pair at all, is not adequate if we want to obtain a minimal ranking function.

Observation 12 *Let P be the set of tolerance-pairs for a first-order knowledge base $\mathcal{KB}$ and a domain D. A minimal model for $\mathcal{KB}$ and D, that is a ranking function for which no world can be assigned a lower rank without loosing the model-property, is not necessarily part of the ranking functions induced by P.*

Minimal tolerance-pairs can be found with a slightly modified version of the algorithm SearchTPair. In algorithm SearchMinimalTPair, only a small part of function test has been changed. In lines 31 to 36, if a tolerance-pair is found, it is checked whether it is smaller or equal to a pair p_{min} already found. If equal, it is added to the set of minimal tolerance-pairs. If it is smaller than the pairs found before, they can be discarded and only the new one is placed in P. In this way, we can ensure that in the end only the minimal tolerance-pairs among all tolerance-pairs are returned. Since all pairs that are stored in set P at the same time must be equal, the pair p_{min} in lines 32 and 34 to perform the comparison can be chosen arbitrarily.

While the adaption described above reduces the return set from all tolerance-pairs to those which are minimal, the added lines 27 and 28 implement an additional pruning strategy. During the search, the pairs in set P are the smallest tolerance-pairs found so far. If the currently processed node represents a partition-pair that is already bigger, its subtree can be safely skipped. The comparison used here, denoted as $>_{PTP}$, is similar to $>_{TP}$ given indirectly by Definition 20, with the only difference that the highest-labeled subsets are not taken into account, since they are not yet final for p and could have additional elements added. That means, a subtree is skipped if the current node's partition-pair already has more tuples than the current minimum or if one subset $p_{\mathcal{R},i}$ or $p_{D,i}$ is smaller than the corresponding subset of the current minimum, with i ranging from 0 to $m-1$. This kind of pruning obviously does not influence the completeness of the search.

Observation 13 *Algorithm* SearchMinimalTPair *is sound and complete, that is, for a first-order knowledge base $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and a domain D, every returned pair is a minimal tolerance-pair and every minimal tolerance-pair that exists is returned.*

Due to the additional pruning of the search tree, algorithm SearchMinimalTPair runs faster than SearchTPair. As a final example in this section, we will look at a

Algorithm 6 SearchMinimalTPair

Input: $\mathcal{KB} = \langle \mathcal{F}, \mathcal{R} \rangle$ and D
Output: Set of minimal tolerance-pairs P

```
1:  P ← ∅, V ← ∅
2:  search(⟨(∅, ∅)⟩, 0, R, D)
3:  function search(p, i, R_L, D_L)
4:      ...
20: function test(p, i, R_L, D_L)
21:     ...
27:     if P ≠ ∅ and p >_PTP p_min ∈ P then          ▷ cannot be a minimum
28:         return false
29:     if not IsPotentialTPair(p, F, R_L) then
30:         return false
31:     if R_L = ∅ and D_L = ∅ then                  ▷ tolerance-pair found
32:         if P = ∅ or p <_TP p_min ∈ P then        ▷ new minimum
33:             P ← {p}
34:         else if p =_TP p_min ∈ P then            ▷ equal to others
35:             P ← P ∪ {p}
36:         return false
37:     return true
```

comparison of the three algorithms for a sample knowledge base. Although the actual runtimes depend on the specific system configuration, their ratios are still informative.

Example 13 At the end of Example 7, we saw that for the knowledge base $\mathcal{KB}_4$ and $D = \{p, t, s, a\}$, there are 43 possible tolerance-pairs. The knowledge base consists of 4 conditionals, 5 predicates and 4 constants, making up 2^{20} possible worlds. Figure 5.4 shows the corresponding runtime for each of the algorithms.

BruteForceTPair generates and tests 2069 partition-pairs to find the tolerance-pairs. SearchTPair only evaluates 462 nodes of its search tree and can find the same tolerance-pairs 22 times faster than BruteForceTPair. The only minimal tolerance-pair for the knowledge base is found by SearchMinimalTPair by evaluating 336 trees nodes, which happens 11 times faster than the search for all tolerance-pairs and 250 times fast than the brute-force search. ■

Algorithm	Partition-Pairs or Tree Nodes	(Minimal) Tolerance-Pairs	Runtime
BruteForceTPair	2069	43	25:09 min
SearchTPair	462	43	1:07 min
SearchMinimalTPair	336	1	0:06 min

Figure 5.4: Comparison of algorithms to create tolerance-pairs

All in all, we can see that the two new backtracking-algorithms presented are a viable alternative to the brute-force approach. They are sound and complete and can run substantially faster. Having all of them implemented in the program demonstrated in Chapter 4, tolerance-pairs can be easily created in order to analyze system Z^{FOL}. With the introduction of an order for the set of tolerance-pairs, preferred minimal tolerance-pairs can now be identified that assign most plausible ranks to possible worlds. However, an overall minimal model, as it is generated by system Z in the propositional case, is not necessarily among the models computed based on tolerance-pairs.

5.5 Evaluation of benchmark examples

In Section 2.5, the idea of testing the behavior of a consequence relation with benchmark examples was introduced. Four central concepts, named property inheritance, irrelevant facts, specificity and exceptional inheritance, were discussed. In this section, we want to evaluate how the consequence relation of the new system, z^{FOL}-entailment, deals with these examples. As it is not yet clear whether system Z^{FOL} can provide a unique ranking function, we look at every tolerance-pairs and its corresponding ranking function. We work with a knowledge base that enables us to test all four concepts in the next example:

Example 14 We extend $\mathcal{KB}_9$ from Example 8. In addition to the common predicates, we introduce E (being an eagle) and R (being red). The domain consists of two individuals $D = \{p, t\}$, the first one is a bird and the second one a penguin. Further, the certain knowledge describes that both penguins and eagles are a subclass of birds. However, there are no conditionals about eagles and there is no information at all about being red.

$$\mathcal{KB}_{11} = \langle\ \{\forall x P(x) \Rightarrow B(x), \forall x E(x) \Rightarrow B(x), B(p), P(t)\}$$
$$\{(F(x)|B(x)), (W(x)|B(x)), (\overline{F}(x)|P(x))\}\ \rangle$$

For this knowledge base, the same two tolerance-pairs p_7 and p_8 exist as shown in Example 8, that differ only by the fact that the wings-conditional is placed in another subset. p_7 is minimal for the knowledge base. The following conclusions can be made with z^{FOL}-

entailment based on the ranking functions computed for p_7 and p_8:

	p_7	p_8
(1)	$E(x) \mid\sim_z^{FOL} F(x)$	$E(x) \mid\sim_z^{FOL} F(x)$
(2)	$P(x) \not\mid\sim_z^{FOL} F(x)$	$P(x) \not\mid\sim_z^{FOL} F(x)$
(3)	$B(x)R(x) \mid\sim_z^{FOL} F(x)$	$B(x)R(x) \mid\sim_z^{FOL} F(x)$
(4)	$P(x) \not\mid\sim_z^{FOL} W(x)$	$P(x) \mid\sim_z^{FOL} W(x)$

This can be easily verified using the implementation, however, the number of possible worlds is too high to give all details here. ∎

As we can see in the example, the consequence relation of system Z^{FOL} provides a reasoning behavior that is similar to that of system Z. Conclusion (1) shows that property inheritance is supported, as the ability to fly, which is given to any bird by the conditional $(F(x)|B(x))$, also applies for eagles. This happens if the subclass relation of birds and eagles is given as certain knowledge, as in the example, or by a conditional, as it would be done in system Z. For the subclass of penguins, two conditionals apply that make a statement about their ability to fly. We expect the system to respect specificity and let the conditional $(\overline{F}(x)|P(x))$ prevail. Hence, it should not be possible to derive that a penguin can fly, and this is indeed shown by conclusion (2). Moreover, (3) illustrates that irrelevant facts do not prevent expected conclusions, as we can, for a red bird, still say that he usually flies, as long as we do not have any information information about red birds being special in any kind.

Conclusion (4), that tests whether exceptional inheritance is supported, shows an interesting phenomenon. With the ranking function computed for tolerance-pair p_7, we cannot derive for a penguin that it usually has wings. This corresponds to the behavior of system Z in the propositional case. However, if we choose the second tolerance-pair p_8, the conditional $(W(x)|(B(x))$ is accepted by the ranking function. The reason is the following: For p_7, the conditional $(W(t)|(B(t))$ is both verified and falsified with rank 0. But, in the ranking function for p_8, every world that falsifies $(W(x)|(B(x))$ has a higher rank, because the conditional is in a subset with a higher index. $(W(t)|(B(t))$ is then falsified with rank 3 and thus is a representative for the conditional. In fact, the same behavior can be achieved in system Z by using a partition of the conditionals in which the wings-conditional is placed in a higher subset than the one it is assigned to by algorithm ConsistencyCheck. However, choosing such a partition conflicts with the aim of assigning the most plausible rank to each world, which would be the result if the original partition is used in system Z.

So far, the benchmark examples show that the reasoning behavior of system Z^{FOL} is the same as in the propositional case. If a minimal tolerance-pair is used to obtain the ranking function for z^{FOL}-entailment, even the drowning problem occurs. The following observation summarizes these results:

Observation 14 *z^{FOL}-entailment handles first-order benchmark examples for property inheritance, specificity, irrelevant facts and exceptional inheritance exactly as z-entailment in the propositional case.*

As another benchmark example, we now look at the popular Nixon diamond, that models a conflict between two conditionals without any of them being more specific. The following example translates it into our first-order setting:

Example 15 We have three predicates, describing that a person is a quaker (Q), a republican (R) and a pacifist (P). For the domain $D = \{n, f\}$, that consists of individuals Nixon (n) and Ford (f), the knowledge base says that Nixon is both a republican and a quaker. The conditionals express that a quaker is usually a pacifist, whereas a republican is usually not a pacifist.

$$\mathcal{KB}_{12} = \langle\ \{R(n), Q(n)\}\ ,\ \{(P(x)|Q(x)), (\overline{P}(x)|R(x))\}\ \rangle$$

For Nixon, obviously both conditionals apply, but express conflicting conclusions. Therefore, we have two possible tolerance-pairs in this case:

$$\begin{aligned} p_1 = \langle\ &(\{(P(x)|Q(x))\},\{f\}),\\ &(\{(\overline{P}(x)|R(x))\},\{n\})\ \rangle \end{aligned}$$

$$\begin{aligned} p_2 = \langle\ &(\{(\overline{P}(x)|R(x))\},\{f\}),\\ &(\{(P(x)|Q(x))\},\{n\})\ \rangle \end{aligned}$$

Though the conditionals are placed differently in p_1 and p_2, Nixon is always in the second subset, as he cannot satisfy one conditional without falsifying the other. In fact, there are only tolerance-pairs because we have a second individual that can serve as a dummy for the other conditional. For $D = \{n\}$ and $\mathcal{KB}_{12}$, no tolerance-pair exists.

If we choose p_1 and compute the ranking function, we can derive that Nixon is not a pacifist, as the formula $\overline{P}(n)$ is accepted, as well as the conditional $(\overline{P}(x)|\top)$ with n as its representative. For p_2, vice versa, $P(n)$ and $(P(x)|\top)$ are accepted. We could say that, in one tolerance-pair, Nixon is a pacifist, and in the other, Nixon is not. An undesired effect occurs if we add additional knowledge about the second individual: By adding $R(f)$ to the facts, we reduce the set of tolerance-pairs to only one, which is p_2, and thereby also "decide" that Nixon is a pacifist. ■

Compared to system Z and the other benchmark examples before, the Nixon diamond is a bit more involved. If we use only one constant, as it seems to be reasonable in this scenario, the conflict between the conditionals manifests in the result that no tolerance-pairs exists at all. By adding a second dummy constant, two tolerance-pairs arise, with each leading to a ranking function that accepts one of the two conclusions that can be

made for Nixon. This is similar to having two different extensions in extension-based systems like Reiter's default logic. However, this result only occurs as long as there is an unspecified constant to resolve the conflict.

At the end of Section 2.5, the concept of preferential and rational consequence relations was introduced and characterized by inference rules. In the last part of this section, we analyze if the consequence relation z^{FOL}-entailment can be categorized by these means. In addition to the inference rules given in Definition 9, a preferential consequence relation can also be induced by a so called *preferential model*, as Kraus et al. show in [KLM90]. Based on such a model, a consequence relation can be defined that satisfies all rules for being a preferential consequence relation. In [LM90], Lehmann and Magidor extend this idea to a first-order language. Their definition of a first-order preferential model, slightly modified to fit the notation used in this thesis, is the following:

Definition 21 (First-Order Preferential Model)
A preferential model W is a triple $\langle S, l, \prec \rangle$ where S is a set, the elements of which will be called states, $l : S \to \mathcal{U}$ assigns a world to each state and $\prec$ is a strict partial order on S satisfying the following two conditions. The first one, the smoothness condition is: $\forall A \in \mathcal{L}_{FOL}$ the set of states $\widehat{A} = \{s \mid s \in S, s \models A\}$ is smooth, where $\models$ is defined as $s \models A$ (read s satisfies A) iff $l(s) \models A$. The second E is:

1. *if a state s, labeled with $\langle M, f \rangle$, is minimal in $\widehat{\exists x A(x)}$, then there exists a state t that is minimal in $\widehat{A(x)}$ and that is labeled with $\langle M, f' \rangle$ where f' differs from f at the most in the element it assigns to x.*

2. *if a state t is minimal in $\widehat{A(x)}$ and labeled with $\langle M, f \rangle$, then there is a state s, labeled with $\langle M, f' \rangle$, where f' differs from f at the most in the element it assigns to x, that is minimal in $\widehat{\exists x A(x)}$.*

With their definition, Lehmann and Magidor try to capture the full scope of first-order logic, for which they need the usual notion of a first-order interpretation. Every element of $\mathcal{U}$, a world, is therefore a pair $\langle M, f \rangle$, with M being a first-order structure, that is an interpretation of function and predicate symbols together with a domain, and f being a function that assigns an element of the domain to each variable. In our more restricted first-order language, we can however represent a world by giving a finite list of the ground atoms that hold, as we did in the previous sections. In order to evaluate the acceptance of conditionals, we do not even need a variable assignment f, as every variable is explicitly or implicitly quantified.

In our approach, a consequence relation is given by a first-order ranking function κ, and based on such a ranking, we can also construct a preferential model. Let S_κ be a set that contains a state for each combination of a possible world $\omega \in \Omega$ and an assignment f of a constant from the domain D to each variable. The function l_κ assigns such a pair $\langle \omega, f \rangle$

to each state. Moreover, let κ define the following order $\prec_\kappa$ over the states: $s \prec_\kappa t$ iff $\kappa(\omega_s) < \kappa(\omega_t)$ for $l_\kappa(s) = \langle \omega_s, f \rangle$ and $l_\kappa(t) = \langle \omega_t, f' \rangle$. That means, the order $\prec_\kappa$ is given by the $<$-relation over integers, which is a strict total order. The smoothness condition is satisfied for every finite set S, as it is the case here, because Ω, D and the set of variables are finite. So finally, we have to show that $W_\kappa = \langle S_\kappa, l_\kappa, \prec_\kappa \rangle$ also satisfies condition E:

1. A closed formula $\exists x A(x)$ is satisfied in every state in which $A(x)$ is true for some constant, independent of the variable assignment in that state. Let s be a minimum among these states with $l(s) = \langle \omega_s, f \rangle$. In this state, we know that $s \models A(a)$ holds for a constant a.
   For the open formula $A(x)$, the set $\widehat{A(x)}$ consists of all states with a world that satisfies $A(x)$ for some constant b and in which this constant b is assigned to x. Hence, $\widehat{A(x)} \subseteq \widehat{\exists x A(x)}$, as both contain the same worlds, but the first set is restricted to specific variable assignments. Then, for the state s, a state t exists in $\widehat{A(x)}$ with the same world ω_s and f' corresponding to f, but assigning a to x. Due to both having the same world assigned, t is also minimal in $\widehat{A(x)}$.

2. Let t be a minimal state in $\widehat{A(x)}$. Since $\widehat{A(x)} \subseteq \widehat{\exists x A(x)}$, this state t is also part of $\widehat{\exists x A(x)}$. And as both sets contain states for exactly the same worlds, with the latter only having additional states for more variable assignments, t must also be minimal in $\widehat{\exists x A(x)}$, as the order only depends on the world assigned to the state.

Therefore, $W_\kappa = \langle S_\kappa, l_\kappa, \prec_\kappa \rangle$ is a first-order preferential model. Such a model induces a consequence relation $\mid\!\sim_W$, that is defined as follows for formulas $A, B \in \mathcal{L}_{\Sigma_{FOL}}$:

$$A \mid\!\sim_W B \text{ iff for any } s \text{ minimal in } \widehat{A}, s \models B$$

Thus, in the minimal states in which A holds, also B must hold. This is in fact the same as requiring that the minimal worlds in which AB holds have a lower rank than the minimal worlds in which $A\overline{B}$ holds. Hence, κ induces a preferential consequence relation.

Observation 15 *Every first-order ranking function induces a preferential consequence relation. As a consequence, also z^{FOL}-entailment is a preferential consequence relation.*

This result, again, corresponds to system Z in the propositional case. In their paper, Lehman and Magidor also discuss rational consequence relations for the first-order case and work towards a notion of rational closure. The fact that the benchmark example with irrelevant facts is handled properly indicates that z^{FOL}-entailment is also a rational consequence relation. And since system Z corresponds to rational closure in the propositional case, it would be interesting to study the relation of a first-order rational closure and system Z^{FOL}. However, at least in [LM90], Lehman and Magidor do not provide an algorithm to construct a rational closure but leave this questions open. The author of this thesis did not find a following paper finishing this work.

5.6 Analysis of first-order properties

In this last section of the chapter, a few final remarks on system Z^{FOL} should be made that have not been mentioned in the previous discussions. Especially, the topics discussed in Section 2.4 will be picked up here and thereby conclude the analysis of the new approach. By moving system Z to a first-order language, the expressiveness of the system greatly benefited. While we can only make statements using propositional variables in system Z, system Z^{FOL} allows us to differentiate between single individuals and classes of individuals and refer to both within a single framework. This makes it possible to model knowledge bases that cannot be expressed with propositional conditionals only.

The restrictions that had to be placed upon a general first-order language have different effects. Only by requiring that the sets of predicates, constants, conditionals and facts are finite, it is possible that an implementation of the system can be provided. The fact that we cannot work with function symbols except constants, reduces the expressiveness of our first-order language, but as we saw in the last section, we can already model many knowledge bases using only predicates and constants. The more severe restriction is that we cannot use predicates with an arity other than one. Examples such as the elephant-keeper-example of Delgrande (cf. [Del98, p. 108]), in which the predicate $L(x, y)$ is used to describe that an elephant likes a keeper, cannot be modeled in system Z^{FOL}.

When we introduced first-order ranking functions in Chapter 3, we already mentioned that every variable in a conditional of system Z^{FOL} is explicitly or implicitly quantified. While the explicit ones are due to inner or outer quantifications in conditionals, an implicit quantification occurs for open conditionals. In the sense of a conditional quantification as described in Section 2.4, system Z^{FOL} lets the open variable range over the set of representatives. Therefore, in contrast to other approaches providing semantics for first-order conditionals, no variable assignments are necessary to evaluate the acceptance of conditionals, as only fully grounded formulas need to be checked for satisfaction.

Another open question is whether system Z^{FOL} provides semantics for first-order conditionals in a subjective or statistical sense. The difference has been discussed in Section 2.4 and depends on a semantics being based on a preference relation among worlds or individuals. Since system Z^{FOL} has a ranking function at the core of its semantics, it seems to be a subjective approach, as it is the case for system Z. However, if we evaluate the acceptance of an open conditional, the concept of representative individuals plays a major role. By dividing the individuals in representatives and non-representatives, a preference among the individuals is brought into the semantics. Although this preference is derived from the ranking over possible worlds, the letter is also derived from a partition of the domain. Hence, in two flavors, a ranking of individuals is also part of the semantics. Overall, this shows that conditionals in system Z^{FOL} have a complex semantic nature with both subjective and statistical aspects, as stated in the next and final observation:

Observation 16 *The semantics of system Z^{FOL}, especially in the case of open conditionals, are a combination of a preference over possible worlds and among individuals.*

Last but not least, we take a look at the lottery paradox. In Section 2.4, we modeled it by the following set of conditionals: $\{\, \forall x(\overline{W}(x)|\top)\,,\, (\exists x W(x)|\top)\,\}$. The final example shows how it can be expressed by the means of system Z^{FOL}.

Example 16 For the sake of simplicity, we look at a small lottery with only two participants $D = \{a, b\}$, though in general, of course, the example is based on the fact that a lottery typically has a high number of participants. The knowledge base has a conditional that says that usually, an individual does not win the lottery. In system Z^{FOL}, this is adequately expressed by an open conditional. For the existential conditional saying that usually, there is one who wins the lottery, we choose a specific constant to denote this individual. That gives us the following knowledge base:

$$\mathcal{KB}_{13} = \langle\, \emptyset\,,\, \{(\overline{W}(x)|\top), (W(a))\}\,\rangle$$

In this case, a tolerance-pair $\langle(\{(\overline{W}(x)|\top)\}, \{b\}), (\{(W(a))\}, \{a\})\rangle$ exists and the corresponding ranking function makes the world in which a wins the lottery and b does not most plausible. However, this result depends on the fact that enough constants are available. If the domain contains only one individual, there is no tolerance-pair. For three constants, three tolerance-pairs arise, leading to ranking functions that prefer worlds in which a always wins and one or both of the others do not win the lottery. ∎

The example shows that we can model the lottery paradox in system Z^{FOL} and do not run into inconsistencies as it is the case in other approaches. Using an open conditional with its both subjective but also statistical semantics, the two statements can be present in a knowledge base. The only limitation is that there have to be enough individuals to provide a representative for each conditional. A similar requirement is also made by the approach of Lehmann and Magidor, and they argue that this is a reasonable condition:

> "We consider [situations] impossible in which there are birds but there is no typical bird. Such situations have been referred to in the literature as the lottery paradox or the bird-shop paradox. For us, such situations are indeed paradoxical. [...] In such situations, we may as well accept that there is a typical bird, perhaps at the cost of accepting that phantom birds exist." [LM90, pp. 62-63]

All in all, we can conclude that system Z^{FOL} is a powerful system that translates the ideas of system Z to a richer language. With first-order logic at hand, knowledge bases can be accurately modeled, using conditionals and certain knowledge as well as variables, constants and quantifications. At the same time, the system provides semantics that cope with first-order issues in a reasonable way.

6 Conclusion

The topic of this thesis was system Z^{FOL}, an approach for first-order default reasoning inspired by its propositional counterpart system Z. In order to achieve its first goal, this thesis provided an implementation that has been created to study and analyze the new system Z^{FOL}. With its interactive graphical user interface, the system can be tested and analyzed. A user can define a knowledge base and let the program create suitable tolerance-pairs for it. Based on such a pair, the corresponding ranking function can be calculated and queries can be evaluated against it. In each step, the program provides the necessary details to follow and understand the computations, and the knowledge bases and results can be imported and exported through a file interface. The program has been implemented based on an existing library for logical knowledge representation and reasoning, and has been designed to be easily extended in the future. Using this program, system Z^{FOL} was studied and the results were presented in this thesis.

First of all, the propositional edge-case of first-order logic was analyzed, coming to the conclusion that system Z is in fact an edge-case of system Z^{FOL}. Although different ranks are assigned to possible worlds by the two ranking functions, they can be transformed into each other and both accept the same conditionals. As a next result, a first algorithm to find tolerance-pairs based on the generate-and-test pattern was presented. It generates every possible partition-pair for a knowledge base and a domain and thereby finds tolerance-pairs. While being sound and complete, this exhaustive approach is not usable for larger knowledge bases. However, based on it, the existing tolerance-pairs for different knowledge bases were studied and a few observations could be derived. Depending on its conditionals, facts and the domain, a knowledge base can have none, one or multiple tolerance-pairs. Since the acceptance of open first-order conditionals is based on representatives, it is crucial to have an individual for each conditional, and different ones for conflicting conditionals. Additional individuals, for which no certain knowledge is given, can make arbitrary tolerance-pairs possible, and therefore, as many facts as possible should be provided for each individual to avoid undesired tolerance-pairs.

In order to provide a better way to find tolerance-pairs, a backtracking search algorithm was developed next. Based on a search tree that is built by constructing partition-pairs step by step, every tolerance-pair can be found. Due to different pruning strategies, only a part of this tree has to be explored in the average case, and the algorithm is therefore superior to the brute force approach. Similarly, it can find every existing tolerance-pair for a knowledge base and a domain. As a next step, inspired by system Z, in which only the minimal partition of the conditionals is used to compute the ranking function, a notion of minimality was introduced for tolerance-pairs. Minimal tolerance-pairs, of which several can exist for a knowledge base, induce ranking functions that assign the most plausible

ranks to worlds compared to other tolerance-pairs. However, there can be more plausible models for a knowledge base that are not induced by a tolerance-pair at all. With only little changes, the backtracking search algorithm could be adapted to search for only these minimal tolerance-pairs. The explored part of the search tree could be further reduced. Therefore, a comparison for a sample knowledge base revealed substantial differences in the runtimes of the three algorithms, with the latter being the fastest.

During the last part of the analysis, the handling of benchmark examples by system Z^{FOL} was analyzed. A difficulty in this task was that there is not only one, but there are possibly several consequence relations for a first-order knowledge base, as it can have several tolerance-pairs and even several minimal among them. For the ranking function computed based on the minimal tolerance-pair of the benchmark knowledge base, the behavior corresponds to that of system Z. The consequence relation sanctioned property inheritance, specificity and could handle irrelevant facts, while suffering from the drowning problem. Further, we saw that every first-order ranking function induces a consequence relation that is a first-order preferential consequence relation. Although system Z^{FOL} imposes some restrictions on the first-order language that is used, it is much more expressive than propositional logic. The semantics given to open conditionals were found to be a weak type of quantification that ranges over typical individuals, called representatives. Therefore, system Z^{FOL} has semantics that are both of a subjective and statistical nature, as worlds but also individuals of the domain are ranked in order to evaluate the acceptance of a open conditional.

With these results, some of the open questions about system Z^{FOL} sketched in Section 3.4 could be answered in this thesis. As of today, there are three different algorithms to find tolerance-pairs for a first-order knowledge base, with a ready to use implementation of them being available. In addition, a notion of minimality for tolerance-pairs has been introduced and studied, paving the way to obtain a unique ranking function as in system Z. Together with the other findings summarized above, this thesis clearly made its contribution to the analysis of the new approach and thereby also achieved its second goal, which was to evaluate system Z^{FOL} and identify its benefits, issues and open questions.

A question that is still open is how we can prove a first-order knowledge base to be inconsistent. With the developed algorithms, we can investigate whether a tolerance-pair exists for a knowledge base or not, as they are complete in this sense. However, if a knowledge base does not have a tolerance-pair, we do not know whether this also implies that it has no model at all. Further, we do not yet know how to obtain a unique ranking function for a first-order knowledge base. The introduced order on tolerance-pairs does not necessarily identify only one tolerance-pair as being minimal, and we therefore can have several ranking functions based on them. Moreover, as we saw during the analysis, the overall minimal ranking function for a knowledge base might not be induced by a tolerance-pair at all. Further works needs to be done to identify which ranking function should be preferred and how it can be computed. A related issue is how a unique consequence relation can be defined for a first-order knowledge base. Although a unique

relation would follow if a unique ranking function were defined, another approach is to define the consequence relation based on all ranking functions of minimal tolerance-pairs. Then, we could make a conclusion only if the corresponding conditional is accepted by each of the ranking functions.

In addition to these open questions, a possible next step for system Z^{FOL} in the future could be an extension to a richer first-order language. While the absence of function symbols might be justifiable, having predicates with higher arities available would make the system even more expressive. Moreover, as there are known shortcomings of system Z in the propositional case, another approach that seems to be worthwhile evaluating is how the successors of system Z, such as system Z^{+} and Z^{*}, can be extended to a first-order language. In [KIT12], this has already been done for c-representations.

Regarding the implementation, several next steps are possible. Although the tool already greatly simplifies the study of system Z^{FOL} compared to a manual evaluation, of course, even more functionalities and interfaces can be added in order to support the work on the open questions. The implemented classes could also be added to the *Log4KR* library, in which ranking functions, system Z and system Z^{FOL} are not yet implemented. A possible approach to cope with the long runtimes for larger knowledge bases would be to incorporate algorithms for satisfiability checking. The algorithm that checks a tolerance-pair, for instance, tries to find a world for each conditional that satisfies this conditional but does not falsify a set of other conditionals. Since possible worlds in our restricted first-order setting are similar to propositional worlds, we could use a so called SAT-solver developed for the propositional satisfiability problem to find such a world without iterating over every possible worlds. However, in order to compute and store a full ranking function in the program, we would still need to generate all possible worlds.

All in all, during the analysis in this thesis, system Z^{FOL} turned out be a powerful system that accomplishes to offer a lot of the features of system Z while at the same time providing a more expressive language to model knowledge. In order to pursue the goal of modeling human reasoning in a formal system, it seems to be an important step forward.

A Appendices

A.1 Syntax for knowledge bases

The following rules are in extended Backus-Naur form according to ISO 14977 and define the expected syntax of a knowledge base that should be parsed by the program:

```
knowledge-base = "signature" , sig ,  "conditionals" , conds , facts ;

sig = [domain] , {predicate , [","]} ;
domain = domain-name , "={" , constant , {"," , constant} , "}" ;
predicate = predicate-name , "(" , [domain-name] , ")" ;

conds = "Conditionals{" , conds-body , "}" ;
conds-body = {conditional , [","]} ;
conditional = "(" , formula , ")" | "(" , formula , "|" , formula , ")" ;

facts = "Facts{" , facts-body , "}" ;
facts-body = {"(" , formula , ")" , [","]} ;

formula = or-expression ;
or-expression = and-expression , {";" , and-expression} ;
and-expression = not-expression , {"," , not-expression} ;
not-expression = "!" , atom | atom ;
atom = predicate-name , "(" , [constant | variable-name] , ")" ;

domain-name = id ;
constant = id ;
predicate-name = id ;
variable-name = u-letter , {letter | digit} ;

id = letter , {letter | digit} ;
digit = "0".."9" ;
letter = l-letter | u-letter | "_" ;
l-letter = "a".."z" ;
u-letter = "A".."Z" ;
```

The domain must be given a name because the parser of library *Log4KR* supports multi-sorted first-order logics. However, in this implementation, only one sort, the domain, is allowed, and every predicate must have arguments of this sort. Further, in general, the parser supports to read multiple named sets of conditionals after `"conditionals"`. In the program of this thesis, in which we work with knowledge bases consisting of conditionals and facts, exactly two sets are expected, one named `"Conditionals"` and one named `"Facts"`. The consequences of the conditionals of the latter are then handled as facts. At any place, the pound sign `#` can start a comment that is ignored by the parser.

A.2 Knowledge bases in import format

In this section, all knowledge bases for system Z^{FOL} used in this thesis are given in the syntax expected by the implemented program. As stated in the next section of the appendix, these knowledge bases are also provided electronically.

Knowledge base 3

```
# KB3
# bear and teddy bear

signature
    D={b,t}
    B(D), T(D), D(D)

conditionals
    Conditionals{
        (D(X)|B(X)),(!D(X)|T(X))
    }
    Facts{
        (B(b)), (T(t))
        (!T(t);B(t)), (!T(b);B(b))
    }
```

Knowledge base 4

```
# KB4
# birds, penguins and super-penguins

signature
    D={p,t,s}
    B(D), P(D), S(D), F(D), W(D)

conditionals
    Conditionals{
        (F(X)|B(X)), (W(X)|B(X)), (!F(X)|P(X)), (F(X)|S(X))
    }
    Facts{
        (B(p)), (P(t)), (S(s))
        (!S(p); P(p)), (!S(t); P(t)), (!S(s); P(s))
        (!P(p); B(p)), (!P(t); B(t)), (!P(s); B(s))
```

```
    }
```

Knowledge base 5

```
# KB5
# propositional edge-case for birds, penguins and super-penguins

signature
    B(), P(), S(), F(), W()

conditionals
    Conditionals{
        (F|B), (W|B), (!F|P), (B|P), (F|S), (P|S),
    }
    Facts{}
```

Knowledge base 6

```
# KB6
# bird and penguin

signature
    D={p,t}
    B(D), P(D), F(D)

conditionals
    Conditionals{
        (F(X)|B(X)), (!F(X)|P(X))
    }
    Facts{
        (!P(p);B(p)), (!P(t);B(t))
    }
```

Knowledge base 7

```
# KB7
# bird and penguin without subclass relation

signature
```

```
    D={p}
    B(D), P(D), F(D)

conditionals
    Conditionals{
        (F(X)|B(X)), (!F(X)|P(X))
    }
    Facts{}
```

Knowledge base 8

```
# KB8
# bird and penguin with facts

signature
    D={p,t}
    B(D), P(D), F(D)

conditionals
    Conditionals{
        (F(X)|B(X)), (!F(X)|P(X))
    }
    Facts{
        (!P(p);B(p)), (!P(t);B(t))
        (B(p)), (P(t))
    }
```

Knowledge base 9

```
# KB9
# bird and penguin with facts and wings

signature
    D={p,t}
    B(D), P(D), F(D), W(D)

conditionals
    Conditionals{
        (F(X)|B(X)), (!F(X)|P(X)), (W(X)|B(X))
    }
    Facts{
```

```
        (!P(p);B(p)), (!P(t);B(t))
        (B(p)), (P(t))
    }
```

Knowledge base 10

```
# KB10
# bird and penguin with facts but no subclass

signature
    D={p,t}
    B(D), P(D), F(D)

conditionals
    Conditionals{
        (F(X)|B(X)), (!F(X)|P(X))
    }
    Facts{
        (!B(t)), (!P(p))
    }
```

Knowledge base 11

```
# KB11
# bird, penguins and eagles

signature
    D={p,t}
    B(D), P(D), F(D), W(D), E(D), R(D)

conditionals
    Conditionals{
        (F(X)|B(X)), (!F(X)|P(X)), (W(X)|B(X))
    }
    Facts{
        (!P(p);B(p)), (!P(t);B(t)), (!E(p);B(p)), (!E(t);B(t))
        (B(p)), (P(t))
    }
```

Knowledge base 12

```
# KB12
# nixon diamon

signature
    D={n,f}
    R(D), Q(D), P(D)

conditionals
    Conditionals{
        (!P(X)|R(X)), (P(X)|Q(X))
    }
    Facts{
        (R(n)), (Q(n)), #(R(f))
    }
```

Knowledge base 13

```
# KB13
# lottery paradox

signature
    D={a,b}
    W(D)

conditionals
    Conditionals{
        (!W(X)), (W(a))
    }
    Facts{}
```

A.3 Contents of the attached CD

The following files are provided on the attached compact disc:

- Electronic version of this thesis (*MasterThesis_Tobias_Falke.pdf*)
- Implemented program (Folder *Program*)
  - Program as an executable JAR-archive (*RelationalSystemZ.jar*)
  - Instructions on how to start the program (*HowTo.txt*)
  - Complete source code (*src/**)
  - Used libraries (*lib/**)
  - Technical documentation as Javadoc (*doc/**)
- Knowledge bases (Folder *KnowledgeBases*)
  - As listed in the preceding section, named *KBx.rcl*

Bibliography

[Ada75] Ernest W. Adams. *The logic of conditionals: An application of probability to deductive logic*. Reidel, Dordrecht, Holland and Boston, 1975.

[Ben90] Bench-Capon, Trevor J. M. *Knowledge representation: An approach to artificial intelligence*. Academic Press, London, 1990.

[BHS93] Wolfgang Bibel, Steffen Hölldobler, and Torsten Schaub. *Wissensrepräsentation und Inferenz - eine grundlegende Einführung*. Vieweg, 1993.

[BKI08] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme - Grundlagen, Algorithmen, Anwendungen*. Vieweg+Teubner Verlag, 4., verbesserte Auflage, 2008.

[Bou99] Rachel A. Bourne. *Default reasoning using maximum entropy and variable strength defaults*. PhD thesis, University of London, 1999.

[Bra97] Ronen I. Brafman. A first-order conditional logic with qualitative statistical semantics. *Journal of Logic and Computation*, pages 777–803, 1997.

[Bre91] Gerhard Brewka. *Nonmonotonic reasoning: Logical foundations of commonsense*. Cambridge University Press, Cambridge and New York, 1991.

[Del98] James P. Delgrande. On first-order conditional logics. *Artificial Intelligence*, 105(1-2):105–137, 1998.

[FHK96] Nir Friedman, Joseph Y. Halpern, and Daphne Koller. First-order conditional logic revisited. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 1305–1312, Portland, Oregon, 1996.

[GMP93] Moisés Goldszmidt, Paul Morris, and Judea Pearl. A maximum entropy approach to nonmonotonic reasoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):220–232, 1993.

[GP96] Moisés Goldszmidt and Judea Pearl. Qualitative Probabilities for Default Reasoning, Belief Revision, and Causal Modeling. *Artificial Intelligence*, 84:57–112, 1996.

[KI01a] Gabriele Kern-Isberner. *Conditionals in Nonmonotonic Reasoning and Belief Revision: Considerung Conditionals as Agents*. Springer, Berlin, 2001.

[KI01b] Gabriele Kern-Isberner. Handling Conditionals Adequately in Uncertain Reasoning. In *Proceedings European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'01*, pages 604–615. Springer LNAI 2143, Berlin, Heidelberg, 2001.

[KIB15] Gabriele Kern-Isberner and Christoph Beierle. A System Z-like Approach for First-Order Default Reasoning. In Thomas Eiter, Hannes Strass, Miroslaw Truszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060, pages 81–95. Springer, 2015.

[KIE14] Gabriele Kern-Isberner and Christian Eichhorn. Structural Inference from Conditional Knowledge Bases. *Studia Logica*, 102(4):751–769, 2014.

[KIT12] Gabriele Kern-Isberner and Matthias Thimm. A Ranking Semantics for First-Order Conditionals. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, pages 456–461. IOS Press, 2012.

[KLM90] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44(1-2):167–207, 1990.

[Leh95] Daniel Lehmann. Another perspective on default reasoning. *Annals of Mathematics and Artificial Intelligence*, 15(1):61–82, 1995.

[Lif89] Vladimir Lifschlitz. Benchmark problems for formal nonmonotonic reasoning. In Michael Reinfrank, Johan de Kleer, Matthew L. Ginsberg, and Erik Sandewall, editors, *Proceedings of the Second international Workshop on Non-monotonic Reasoning*, pages 202–219, New York, NY, USA, 1989. Springer.

[LM90] Daniel Lehmann and Menachem Magidor. Preferential Logics: the Predicate Calculus case: (extended abstract). In *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 57–72, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[LM92] Daniel Lehmann and Menachem Magidor. What does a Conditional Knowledge Base Entail? *Artificial Intelligence*, 55(1):1–60, 1992.

[Łuk90] Witold Łukaszewicz. *Non-monotonic reasoning: Formalization of commonsense reasoning*. Ellis Horwood, New York, 1990.

[MS08] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer-Verlag, Berlin, Heidelberg, 2008.

[Pea90] 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.

[Spo88] Wolfgang Spohn. Ordinal Conditional Functions: A Dynamic Theory of Epistemic States. In William L. Harper and Brian Skyrms, editors, *Causation in Decision, Belief Change, and Statistics*, volume 42, pages 105–134. Springer, 1988.

Statement of Authorship

I declare that I have written this thesis on the topic

Computation of ranking functions for knowledge bases with relational conditionals

independently and without inadmissible usage of third parties. I used only the specified sources and tools and marked all parts that are taken literally or correspondingly from these sources according to scientific standards. This assurance of independent work also applies to drawings, sketches and figures. The thesis has not been presented in this or a similar form to any examination authority before and has not yet been published. By the submission of the electronic version of the final thesis, I notice that it is checked using a plagiarism detection service and kept only for examination purposes.

Mannheim,
April 19, 2015