A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

SIKS Dissertation Series No. 2004-03
The reseach reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN: 90-9017799-X

VRIJE UNIVERSITEIT

A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op dinsdag 23 maart 2004 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Petrus Cornelis Groot

geboren te Alkmaar

promotor:	prof.dr. F.A.H. van Harmelen
copromotor:	dr. A.C.M. ten Teije

Epigraph

The Blind Men and the Elephant

It was six men of Indostan
To learning much inclined,
Who went to see the Elephant
(Though all of them were blind),
That each by observation
Might satisfy his mind

The First approached the Elephant,
And happening to fall
Against his broad and sturdy side,
At once began to bawl:
"God bless me! but the Elephant
Is very like a wall!"

The Second, feeling of the tusk,
Cried, "Ho! what have we here,
So very round and smooth and sharp?
To me 'tis mighty clear
This wonder of an Elephant
Is very like a spear!"

The Third approached the animal,
And happening to take
The squirming trunk within his hands,
Thus boldly up and spake:
"I see," quoth he, "the Elephant
Is very like a snake!"

The Fourth reached out an eager hand,
And felt about the knee.
“What most this wondrous beast is like
Is mighty plain,” quoth he;
“’Tis clear enough the Elephant
Is very like a tree!”

The Fifth, who chanced to touch the ear,
Said: “E’en the blindest man
Can tell what this resembles most;
Deny the fact who can
This marvel of an Elephant
Is very like a fan!”

The Sixth no sooner had begun
About the beast to grope,
Than, seizing on the swinging tail
That fell within his scope.
“I see,” quoth he, “the Elephant
Is very like a rope!”

And so these men of Indostan
Disputed loud and long,
Each in his own opinion
Exceeding stiff and strong,
Though each was partly in the right,
And all were in the wrong!

Moral:

So oft in theologic wars,
The disputants, I ween,
Rail on in utter ignorance
Of what each other mean,
And prate about an Elephant
Not one of them has seen!

—John Godfrey Saxe

Contents

Preface

And finally, at the end of writing this dissertation, I can reflect on my work and time as a PhD student and acknowledge those people that contributed in the process. First of all there are my supervisors. Actually, I started with one, which became none, and ended up with two.

My first year as a PhD student was in the theoretical computer science department, which didn't work out because after one year my supervisor left the university. The group I worked in was nice, but I did not mind it too much, because a new position became available in the group of Frank van Harmelen. We already knew each other from the time I was a student, and he was eager to accept me as his PhD student.

Eager is just the term that describes Frank. Frank always seems to have plenty of energy and does not seem to need any sleep. Don't be surprised if you receive an answer to your e-mail three o'clock in the morning. His energetic presence is also a very motivating force for the people around him.

As Frank closely collaborated with Annette ten Teije, it was quickly decided that she would also help in supervising my dissertation. The good qualities I like about her is her thoroughness and her preciseness. Definitely useful when receiving feedback.

Furthermore, I like to thank some of my fellow PhD students. First of all Nam Kyoo, who I already knew before coming to the Vrije universiteit. He was the one that motivated me to also study Mathematics besides studying AI. A choice I'm glad to have made, although he told me on multiple occasions he thought otherwise after I received my first grade. Then there is Sandjai, who always seems to have an answer to any question I pose him. I'm glad he returned to the Vrije universiteit before I finished my dissertation. Ofcourse, this acknowledgement wouldn't be finished without thanking the 'hackers' Jason, Rob, and Ronald, who I met when I became a student and also stayed for a PhD position. And finally a thanks to Martine for her tea breaks, although it was disappointing to hear her say that bridge was not something for her to pursue (but at least she tried).

Bridge was sometimes played during lunch time. Something I enjoy doing and will probably continue to do as long as I can hold the cards. Therefore, thanks to all those people that participated. I won't mention all their names because there are many and I would risk forgetting someone.

I will close this preface with a final remark about the cover. The inspiration for the cover comes from a poem ‘The Blind Men and the Elephant’ written by the American poet John Godfrey Saxe (1816–1887). The poem is based on a fable which was told in India many years ago. The poem can be found in the epigraph of this dissertation. It is a nice example that shows that filling in blanks, because of incomplete or incorrect sensory data, is definitely non-trivial and can easily lead to misinterpretations. In particular, using approximation in symbolic problem solving, the topic of this dissertation, is a challenging subject for further research.

Chapter 1

Introduction

1.1 Research goal and context

This dissertation addresses the following *research goal*:

> "A theoretical and empirical analysis of the effect of approximation on symbolic problem solving."

This research goal contains a number of concepts that are clarified in this introductory section.

The first concept in the research goal is the *use of approximation*, i.e., in dictionary terms something that is fairly accurate but not totally precise. In particular, this dissertation focusses on approximation that is used by programs used in the field of symbolic problem solving. Those programs return approximate solutions that are fairly accurate but not totally precise when compared to optimal solutions. Furthermore, this dissertation focusses on approximation already present in currently existing methods and approximation obtained by applying existing approximation techniques to new domains of interest.

The second concept in the research goal is the *effect* of approximation for solving problems. There has to be a reason to prefer approximating methods that produce sub-obtimal solutions instead of methods that produce optimal solutions. In this dissertation such an effect is focused on when analyzing approximation. Several reasons can be given that support the use of approximation, not just in the field of symbolic problem solving, but also in many other disciplines, like Artificial Intelligence, Computer Science, Mathematics, and Economics. Three of those reasons are discussed in more detail:

Reducing computational complexity: Problems exist that cannot be computed in a reasonable time period. For example, suppose we have a problem that consists of $n+1$ objects and to solve the problem requires twice as much time as solving the problem with n objects. If the problem with 1 object could be solved in 1 second, then solving the problem with n objects would take $2^{(n-1)}$ seconds. The problem with 13 objects would not be solvable within a hour, with 18 objects it would not be solvable within a day, and with 26 objects it would not be solvable within a year. For n a number could even be chosen such that the problem would not solvable even if someone had worked on it without waisting a single minute since the creation of mankind. An example of a problem that exhibits this kind of behaviour is the 'Tower of Hanoi' problem, which is discussed in more detail in Section 7.3.1.2. Note that even if a problem could be solved exactly in a reasonable time period it might not do us any good. Situations occur in our everyday lives that need to be processed before a certain deadline, e.g., publishing a newspaper, preventing a nucleair meltdown, or administering medication to a (dying) patient. Even if the perfect solution could be computed, obtaining it after the deadline has expired will make it useless. For some problems, such a deadline may not always be known beforehand. Approximation allows us to make a trade off between computation time and solution quality: a reasonable solution within reasonable time instead of the perfect solution (or no solution) in an unreasonable amount of time.

Controlling the number of solutions: A method used to solve a problem might either return too many solutions, too few, or no solution at all. For some problems this may be unsatisfactory. For example in diagnosis, giving a number of treatments to a patient from which he may choose one or follow them all at once is usually not a good idea. Some of the treatments might conflict with the other treatments, are more expensive or more hazardous for the patient, or address a disease that is unlikely to be present. Instead of returning many treatments only one (or a few) should be offered to the patient. Obviously, telling a patient "to sit this one out and see what will happen", i.e., offering no explanation or treatment, may be just as unsatisfactory. Instead of giving the responsibility of reducing or increasing the number of solution to for example a docter, one can also try to restate the original problem in another form, e.g., by adding or removing restrictions. This will lead to an approximate problem, which should allow for more or less solutions that approximate the solutions to the original problem. Hence, approximation can be used to control the number of solutions to a problem. (In Chapter 9 a method that can be used to increase or decrease the outcome of any logical inference problem is described in more detail.)

Inherent approximation: Many systems, and in particular those found in symbolic problem solving, make use of knowledge and data input given by a user or another system. It is unreasonable to assume that these elements are always perfectly represented. Usually there are errors present in the data or knowledge or they may be incomplete. There are many reasons that may lead to incorrect or incompete data and knowledge. For example, errors may occur in the data, i.e., the observed quantity differs from its true value, because the measuring device used is imprecise (e.g., our eyes). Data may be incomplete, because to obtain them would be too costly or too risky. Knowledge may be incorrect or incomplete because of limitations of the representation used by the system, because the knowledge of an expert has been incorrectly interpreted by the system designer, or because the knowledge of the expert is faulty. Note that even among experts there is not always a consensus about what is true or false in their domain of expertise. Approximation is therefore inherently present in many problems. Studying approximation may lead to a better understanding of these problems.

Either one or a combination of all the above mentioned reasons can underly the choice for using or analyzing approximations. These reasons occur in many areas and the use of approximation occurs in many fields. This dissertation focusses mainly on the use of approximation in symbolic problem solving.

The third concept in the research goal is the *restriction to problems in symbolic problem solving*. A typical architecture for solving problems in symbolic problem solving is shown in Figure 1.1. It consists of a reasoning method (i.e., algorithm), a knowledge base, and an input, which together produce an output. For example, in diagnosis the knowledge base consists of rules and facts about the way some system is expected to behave, while the input consists of the observed behaviour of the system. The reasoning method can be some form of logical inference that gives a diagnosis as output, which gives an explanation whenever a discrepancy is found between the observed behaviour and the expected behaviour.

In this dissertation, various forms of approximation will be analysed. All these forms of approximation can be applied to problems that fit the architecture of Figure 1.1. This architecture can therefore be considered to be the underlying framework that links the various chapters to each other. More specifically, this dissertation is divided into three parts. Part I describes a form of approximation related to the knowledge base and the data input. Part II describes a form of approximation related to the knowledge base. Part III describes a form of approximation related to the reasoning method.

Besides this typical architecture there are a number of characteristics typical for problems in symbolic problem solving when one wants to exploit approximation for solving such problems. These characteristics are the following:

Use of logic: Many symbolic problem solving systems use some form of logic as representation and logical inference to derive a solution. Hence, we are not dealing with numerical problems. This means there is no obvious metric that tells us "how far we are" from the right answer to an inference problem.

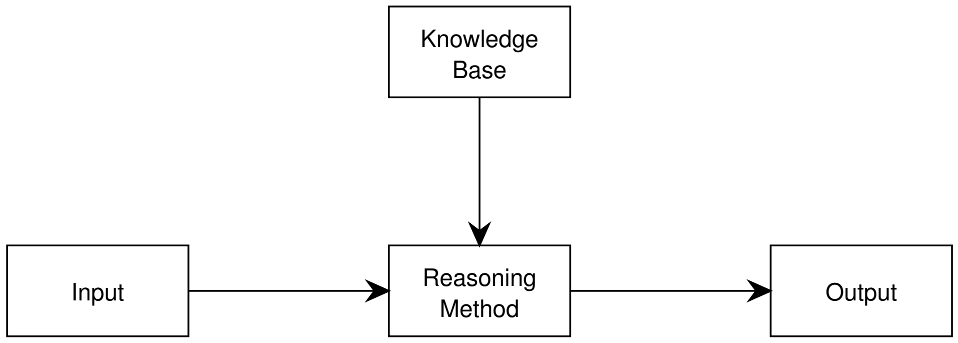


Figure 1.1: A typical architecture for solving problems in symbolic problem solving.

Multiple forms of approximation: Problems in symbolic problem solving can usually be solved by approximations in more than one way. A problem that fits the architecture in Figure 1.1 might be approximated in *three* different ways: the reasoning method, the knowledge base, or the input. Either one of the components can be approximated, or some combination of approximations can be used.

Inherent presence of approximation: Symbolic problem solving systems usually use some knowledge about a domain or system. This knowledge is a model of something in the real world, and therefore already an approximation. Hence, approximation is already inherently present in many symbolic problem solving systems.

All concepts contained in the research goal stated at the beginning of this section have now been discussed in more detail. However, the research goal is formulated quite generally. Section 1.2 discusses in more detail our approach in answering the research goal by looking at three specific areas that fit the research goal. The rest of this chapter gives a background of concepts and methods related to the research in this dissertation (Section 1.3), the contributions of this dissertation (Section 1.4), and an overview of the structure of the dissertation (Section 1.5).

1.2 Approach

The approach taken in this dissertation is to analyse three areas in particular in which approximation is already present or can be used in a promising way to solve a particular problem. These areas are the following:

I. Knowledge Based Systems.

II. Knowledge Compilation.

III. Approximate Entailment.

In the area of KBSs approximation is always present because of the approximate nature of knowledge and data (inherent approximation). Practice has proven that KBSs are an important area in the field of symbolic problem solving. However, although KBSs are claimed to be robust to incorrect and incomplete knowledge or data, little or no attempt has been made to analyse and quantify this property of KBSs. In the area of knowledge compilation the methods are claimed to be ready for practical use. Nevertheless, little is reported on case studies and empirical results. Similar considerations also apply to the area of the approximate entailment operator developed in [Schaerf and Cadoli, 1995]. From a theoretical point of view, the method satisfies some properties desirable for an approximation method, yet little is reported on the practical usefulness of such a method.

This dissertation therefore looks at the following goals in more detail:

1. Quantifying the robust behaviour of KBSs in the presence of incorrect and incomplete knowledge or data.

2. Analyse the applicability of knowledge compilation techniques to planning problems.
3. Analyse the applicability of the approximate entailment operator to classifcication problems.

These three specific goals all contribute to the more general goal of analysing the effect of approximation on solving problems in symbolic problem solving.

1.3 Related work

There is a large amount of literature dealing in some way with approximation. It falls outside the scope of this dissertation to describe this vast amount in detail. This section is limited to those concepts that have a major influence in the field of symbolic problem solving and those that are related to this dissertation. This related work section is divided according to Figure 1.1 as this is the underlying structure that connects the various topics discussed in this dissertation. Section 1.3.1 discusses approximations related to the reasoning method, Section 1.3.2 discusses approximations related to the knowledge base, and Section 1.3.3 discusses approximations related to the input.

1.3.1 Approximations related to the reasoning method

Approximating the reasoning method to solve a problem approximately is probably the most well known form of approximation among the forms of approximation identified in Figure 1.1. A simple example of an approximation algorithm is demonstrated in the following two player game. Given an interval $[a, b]$, either continuous or discrete, and two players A and B, player A picks a number n from the interval $[a, b]$ and player B has to guess it. Player B may repeatedly pick any number m from the interval $[a, b]$ and player A will tell him if $n < m$, $n > m$ or $n = m$ holds. An approximation algorithm for player B would be to repeatedly pick a number from the remaining interval that contains n.

Although simple, this algorithm belongs to an important group of approximation alghorithms called ‘anytime algorithms’. Anytime algorithms are algorithms that exchange execution time for quality of results. The term anytime algorithm was coined by Dean and Boddy in the late 1980’s [Boddy and Dean, 1989, Dean and Boddy, 1988] in their work on time-dependent planning. A similar idea, called flexible computation was introduced in [Horvitz, 1987] in 1987 in solving time-critical decision problems.

Anytime algorithms are important for symbolic problem solving for two reasons. First, although many problems require a lot of resources (e.g., time) to solve them, many systems can already produce good partial solutions in a short amount of time. A system that can reason about how much time is needed to obtain an adequate result may be more adaptive in complex and changing environments. Second, a technique for reasoning about allocating time need not be restricted to the internals of a system. Intelligent agents must be able to reason about how fast they and other agents can manipulate and respond to their environment.

Not every algorithm that trades execution time for quality of results is necessarily an anytime algorithm. The properties desirable for anytime algorithms are the following [Zilberstein, 1996]:

Measurable quality: The quality of an approximate result can be determined precisely.

Recognizable quality: The quality of an approximate result can easily be determined at run time.

Monotonicity: The quality of the result is a nondecreasing function of time and input quality.

Consistency: The quality of the result is correlated with computation time and input quality.

Diminishing returns: The improvement in solution quality is larger at the early stages of the computation, and it diminishes over time.

Interruptibility: The algorithm can be stopped at any time and provide some answer.

Preemptability: The algorithm can be suspended and resumed with minimal overhead.

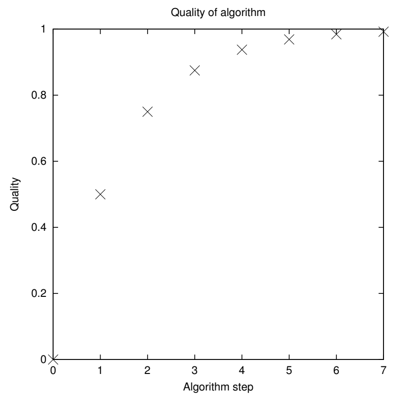


Figure 1.2: Quality of example algorithm for first seven steps.

The alghorithm described in the beginning of this section satisfies these properties. The result of the algorithm after each step is the smallest remaining interval that contains the number n we seek. By repeatedly choosing a number from the interval that contains n, it will become smaller and therefore be a better approximation of n. Let us denote the interval given as output by the algorithm by I, its length by $l(I)$, and define the quality of the algorithm by

$$Q = \frac{(b-a) - l(I)}{(b-a)}.$$

Then Q is a non-decreasing function which can easily be computed. In fact, when we divide the interval each time in halve and the number n has not be found yet, the quality of the algorithm can be computed exactly beforehand without running the algorithm and can graphically be represented as is done in Figure 1.2. This graph clearly demonstrates some of the desired properties (e.g., monotoniciy, diminishing returns, etc.). A graph like Figure 1.2 in which the quality of an algorithm is plotted against execution time is called a *performance profile*.

Since the work of Dean and Boddy [Boddy and Dean, 1989, Dean and Boddy, 1988], the context in which anytime algorithms have been applied has broaden from planning and decision making to include problems from sensor interpretation to database manipulation, and the methods for utilizing anytime techniques have become more powerful. In 1991, S.J. Russel and S. Zilberstein completed work on composing anytime algorithms into more complicated systems [Russell and Zilberstein, 1991, Zilberstein, 1993, Zilberstein and Russell, 1996]. By proving composition could be optimally performed (i.e., using information about the algorithm's performance to determine the best way to divide the time between the components), it became possible to build complex anytime systems by combining simple anytime components.

Section 1.1 stated that a characteristic of systems in symbolic problem solving is the use of logic. This section looks at some approximate reasoning methods that can be used to approximate any logical inference problem that uses the logical entailment operator.

One of those methods is Boolean Constraints Propagation (BCP) which is a variant of unit resolution [McAllester, 1990]. BCP is a sound, but incomplete linear-time reasoner that can be used to approximate the logical entailment relation. Sometimes BCP is also called *clausal BCP*, because BCP is usually restricted to clauses. This restriction makes BCP tractable. If BCP is not restricted to clauses, but general formulas are allowed, the reasoner is called *formula BCP* and the method becomes intractable.

In [deKleer, 1990] two techniques are suggested for using clausal BCP for theories containing arbitrary formulas. In one, *CNF-BCP*, the formulas are first converted into CNF, and in the other, *Prime-BCP*, clausal BCP is applied to the prime implicants of each formula. Since computing prime implicants is itself an intractable problem, Prime-BCP is also inherently intractable.

For CNF-BCP there are two methods to transform the formulas into CNF. If no new symbols are added, then the conversion to CNF may lead to an exponential increase of the size of the given theory. The transformation of a theory to CNF can be done in linear time and space if new symbols are allowed to be added to the theory [Cook, 1971]. Each new symbol will be used to represent some sub-formula of the theory. However, with this method, reasoning with CNF-BCP is strongly inhibited.

Another method that extends BCP to non-clausal theories is Fact Propagation (FP) [Dalal, 1992]. FP can be specified using a confluent rewrite system, for which an algorithm can be written that has quadratic-time complexity in general, but is still linear-time for clausal theories. Another advantage of FP is that it sometimes performs more inferences than CNF-BCP. A restricted form of FP (RFP) also exists which infers exactly the same literals as CNF-BCP.

All the discussed methods (i.e., BCP, FP, and RFP) are incomplete reasoners. In [Dalal, 1996a, Dalal, 1996b] a general technique is presented that can extend any incomplete propositional reasoner satisfying certain properties to a family of increasingly-complete, sound and tractable reasoners. Such a family of increasingly-complete, sound and tractable reasoners is called an 'anytime family' of propositional reasoners, which, as the name implies, can be used as an anytime algorithm.

Another method for approximating logical inferencing was devised by Cadoli and Schaerf and is called S-1- and S-3-entailment [Schaerf and Cadoli, 1995], which is described in more detail in Chapter 9. This method uses a semantical approach and is based on a 'classic' method for incomplete reasoning, which has been introduced by Levesque [Levesque, 1984, Levesque, 1988, Levesque, 1989] and has since been studied by many other authors.

The method of Cadoli and Schaerf allows both sound approximations and complete approximations and the approximate answers can be improved when more resources (e.g., computation time) is given to the reasoner. The approximate answer will converge to the right answer provided there is enough time and motivation. Summarizing, the method of S-1- and S-3-entailment fulfill the following guidelines, that are proposed by Cadoli and Schaerf to be desirable for any approximation method:

Semantically well-founded: Approximate answers should give semantically clear information about the problem at hand.

Computationally attractive: Approximate answers should be easier to compute than answers to the original problem.

Improvable: Approximate answers can be improved, and eventually they converge to the right answer (provided there is enough time and motivation).

Dual: Both sound approximations and complete ones should be described.

Flexible: The approximation method should be general enough to be applicable to a wide range of reasoning problems.

Some applications of this method to approximate diagnosis are reported in [tenTeije and vanHarmelen, 1997, tenTeije and vanHarmelen, 1996]. Chapter 9 discusses the method in more detail and analyzes its applicability to classification problems.

1.3.2 Approximations related to the knowledge base

Considering the three reasons stated in Section 1.1, two of those reasons apply in the context of the knowledge base. These reasons are reducing the complexity of inferencing from a knowledge base, and the inherent approximate nature of knowledge stored in a knowledge base.

A well known problem in symbolic problem solving is the computational complexity of reasoning. An area that deals with this problem for knowledge bases written in some logical language is 'knowledge compilation'. the underlying idea of knowledge compilation is that a knowledge base does not change much over time. The goal of knowledge compilation is to translate the knowledge into (or approximate by) another knowledge base with better computational properties. This 'compiled' knowledge base can be re-used to solve many problem instances, thereby saving time and computational resources when compared to solving the problem instances with the original knowledge base. In Part II of this dissertation we look at knowledge compilation techniques for speeding up solving planning problems. Details of the ideas underlying knowledge compilation and of the methods that belong to this research area will be deferred to Chapter 5.

1.3.3 Approximations related to the input

With input any form of data is meant that is given to a system at runtime by either a user or another system. Examples are observations, sensor data, database queries, etc. All three reasons mentioned in Section 1.1 for using or analyzing approximations are applicable to the input. First, reducing the complexity of a problem may be done by simplifying its input. For example a query may be split into subqueries or may be approximated by another query to reduce the complexity of computing an answer to the query. Second, the input can be adjusted to change the number of solutions to the problem. For example, a user who poses a query to a search engine on the WWW and receives 10.000 hits (i.e., matching pages) may use this knowledge to adjust his original query into one that reduces the number of hits. Third, input may be incomplete and incorrect.

1.4 Contributions

For each of the three areas explored, I. Knowledge-Based Systems, II. Knowledge Compilation, and III. Approximate Entailment, the focus will be on a particular research goal (Section 1.2). Time spent on these research goals lead to new insights, extensions of existing methods, and the development of new methods. The contributions of this dissertation are summarized below.

- I.1 Methodology for measuring the robustness of KBSs.
- I.2 Application of the proposed methodology to a particular KBS.
- II.1 Empirical analysis of the use of knowledge compilation for planning problems.
- III.1 A formal analysis of classification, including forms of approximate classification.
- III.2 A formal analysis of the use of approximate deduction for classification reasoning.
- III.3 Empirical analysis of approximate classification through approximate deduction.

This list serves as an appetizer. At the end of this dissertation (Section 11.1), after the research behind these contributions has been presented, these contributions will be described in more detail.

1.5 Structure of this dissertation

Part I: Quantitative Quality Measures. KBSs model the knowledge about some part in our world. Models are usually not exact, but only an approximation. This is in particular true for KBSs as the knowledge they represent or the data they use can be incorrect and incomplete. Nevertheless, KBSs containing errors can still function at acceptable levels. How well KBSs can deal with incomplete and incorrect data and knowledge is an essential dimension of their validation. In this part a general approach is proposed to provide a general setting for quantitative quality measures for KBSs. To show the feasibility of the approach, the approach is applied on a large and realistic vegetation-classificaton system.

Chapter 2 Introduction: This chapter gives the motivation for quantitatative analysis of KBSs. It introduces and formalizes the methodology one can use for such quantitative analysis.

Chapter 3 The degradation studies: Using the methodology and KBS described in the previous chapter, this chapter describes the experiments performed with them. The experiments includes a robustness analysis with respect to incomplete input, an incomplete knowledge base, and an incorrect knowledge base.

Chapter 4 Final words: This chapter looks at other uses of the proposed methodology, gives conclusions based on the experimental results, and looks at future work.

Part II: Knowledge Compilation. Many systems in symbolic problem solving use a logic language for representation, which has a number of benefits. However, using a logic language also has a serious drawback: the computational complexity of inference. Over the years many techniques have been devised to deal with this problem. One of those techniques is knowledge compilation. In knowledge compilation the part of the theory that remains constant over many problem instances is translated into (or approximated by) another theory with better computational properties. The goal is to speed up the actual problem solving by using the compiled theory instead of the original theory. This part of the dissertation investigates the applicability of knowledge compilation techniques to a number of planning problems.

Chapter 5: Knowledge compilation. Describes knowledge compilation and gives an overview of a number of compilation techniques.

Chapter 6: Planning. Defines the domain of planning problems. Shows how a planning problem can be solved as a satisfiability problem. The rest of the chapter deals with various encodings that can be used to encode a planning problem into propositional logic.

Chapter 7: Applying knowledge compilation. This chapter starts by explaining how knowledge compilation can be applied to planning problems. Thereafter it describes the choices one has to make for applying knowledge compilation to a planning problem: the planning problem, the translation of the planning problem, and the knowledge compilation method. After some motivated choices, the rest of the chapter shows the experimental results.

Part III: Approximating Classification. Object and class descriptions need not always be represented completely or without errors. For example, class descriptions may have been constructed from incomplete and/or incorrect information sources like system hierarchies or bookmark folders. Classification methods need therefore be robust enough when dealing with such less than ideal information sources. In this part classification is formalized and approximated using a general approximation technique.

Chapter 8: Classification. Describes classification, identifies various criteria and various representation choices, and gives a logical formalization of all obtained variations of classification. The section concludes with two approximate forms of classification.

Chapter 9: Approximating classification. Describes a general method developed by Cadoli and Schaerf for approximating the logical entailment relation. This method is applied to classification and the obtained approximations are analysed using the rules of propositional logic and properties of the approximate entailment operator.

Chapter 10: Empirical analysis. As a theoretical analysis does not provide all answers for an effective use of the approximate entailment operator, this chapter looks at empirical methods for obtaining information that extends (or supports) the results of the theoretical analysis of Chapter 9.

Part IV: Contributions and Discussion. The contributions of the dissertation are summarized and the dissertation as a whole is reflected upon.

Part I

Quantitative Measures

Chapter 2

Introduction

When asked about the essential differences between Knowledge-Based Systems (KBSs) and 'conventional software', one often hears the claim that KBSs can deal with incomplete, incorrect and uncertain knowledge and data, whereas conventional software is typically very brittle in these respects (see e.g., [Hayes-Roth, 1984] for a very early formulation of this claim). Although, nowadays researchers no longer view this distinction as either necessary or sufficient to define a KBS, it is believed that the ability of KBSs to deal with missing or invalid data is an essential dimension of KBS validation.

There has been both practical experience and theoretical analysis over many years to back up the mentioned claim. As an example of practical experience, [Preece *et al.*, 1997] reports that in a number of verification exercises, errors were found in the knowledge-base of KBSs which were nevertheless still functioning at acceptable levels. As an example of theoretical analysis, [tenTeije and vanHarmelen, 1996] and [tenTeije and vanHarmelen, 1997] prove that for a large class of diagnostic systems the computed set of diagnoses degrades gracefully and predictably when either the system input (observations) or the knowledge-base degrades in quality.

However, until now, the analysis of the robustness of KBSs in the face of incomplete, incorrect or uncertain knowledge and data has been limited to such practical experience and qualitative analysis. Little or no attempt has been made at a quantitative analysis of the proclaimed robustness of KBSs. A recent special issue of a journal was dedicated to methods for evaluating Knowledge-Based Systems [Menzies and vanHarmelen, 1999]. None of the papers in that special issue performed any quantitative analysis on the quality of Knowledge Based Systems. The editorial of this special issue lists only a hand-full of quantitative evaluation studies that have been performed over a decade or more of KBS research. In fact, one paper in that special issue [Shadbolt *et al.*, 1999] even seems to suggest that global qualitative evaluations are about as much as we can expect from KBS evaluation projects. Finally, one of the reviewers of this paper even remarked that 'for a long time, the KA community has decried the lack of good evaluation metrics to measure the quality of the KA process and of the resulting knowledge bases.' We consider this a serious defect in the study of KBSs, particularly since such robustness is often proclaimed as a unique characteristic of KBSs.

The aim of this research is to show that *a quantitative analysis of the robustness of*

KBSs is both possible and useful. To argue this claim, we present a case study in which we perform such a quantitative analysis for a particular KBS. In Section 2.1 we describe our approach to measuring robustness by degradation studies, and we give definitions for the basic notions involved in such degradation studies. In subsequent sections, we apply this approach in a case study. Section 3.1 describes the KBS which we subjected to a degradation study. Section 3 gives an overview of the degradation studies we performed. Thereafter, Section 3.3 reports our robustness results with respect to the data inputted and Section 3.4 the robustness results with respect to the knowledge base used. The results obtained will be analysed in these two sections. Finally, in Section 4.2 summarizes the main points of the paper and we look at future steps to be taken.

2.1 Approach and foundational definitions

In this section we describe our approach to measuring robustness by degradation studies, and we give definitions for the basic notions involved in such degradation studies. Our aim is to define a very general set of notions that can be widely used in future degradation studies. We regard this section as the central contribution of this research: the definitions in this section should form the basis of similar analyses by other researchers and practitioners.

2.1.1 Robustness and degradation

The IEEE Standard Glossary of Software Engineering Terminology [IEEE, 1990] gives the following definition for robustness:

Informal Definition 2.1.1.1 (Robustness) *The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.*

In other words, robustness of a KBS is concerned with the way in which the quality of the KBS output degrades as a function of a decrease in the quality of the KBS input. This definition immediately leads to the idea of degradation studies:

Informal Definition 2.1.1.2 (Degradation Study) *In a* degradation study *we gradually decrease the quality of the KBS input, and measure how the KBS output quality decreases as a result.*

This informal definition contains the concepts of 'KBS input' and 'output quality' which we will discuss in more detail in the Sections 2.1.2 and 2.1.3.

2.1.2 Output quality

Of course, we must be more precise about the rather vague notion of 'quality' of the KBS input and output. Concerning the KBS output, we assume that this is always a *set* of answers. In fact, for many typical KBS tasks, this is a realistic assumption: a set of consistent classes in a classification task, a set of likely hypotheses in a diagnostic task, a set of potential designs in a configuration task, etc.[i] More explicitly stated:

Assumption 2.1.2.1 *For the KBSs that we consider we assume that their output can be interpreted as a discrete set of answers.*

Under this assumption, we define two measures for KBS output quality. Let *correct(I)* be the set of all correct answers for a given input I, and *output(I)* be the set of actually computed answers for a given input I.

Definition 2.1.2.2 (Recall) *The* recall(I) *of a KBS for a given input I is defined as:*

$$recall(I) = \frac{|correct(I) \cap output(I)|}{|correct(I)|}.$$

In other words: the recall is the fraction of correct answers that the system actually computes. It can of course happen that $correct(I) = \varnothing$, i.e., there is no correct answer. For example, this happens when the system is presented with a case I for which no correct output exists, such as an inconsistent set of observations for a classification system, or an inconsistent set of requirements for a design system. In this case we define:

$$\text{if } correct(I) = \varnothing \text{ then } recall(I) = 1,$$

as all correct answers are recalled by the system.

Definition 2.1.2.3 (Precision) *The* precision(I) *of a KBS for a given input I is defined as:*

$$precision(I) = \frac{|correct(I) \cap output(I)|}{|output(I)|}.$$

In other words: the precision is the fraction of computed answers that are actually correct. In the case $output(I) = \varnothing$ (i.e., the system returns no output) we define:

$$\text{if } output(I) = \varnothing \text{ then } precision(I) = \begin{cases} 1 & \text{if } correct(I) = \varnothing, \\ 0 & \text{otherwise.} \end{cases}$$

This reflects the intuition that the only correct output in this case is no output at all.

Ideally, both recall and precision would be 1 (i.e., the system returns all and only correct answers), but in practice they are antagonistic: a higher precision (recall) is usually paid for with a lower recall (precision).

There are two attractive aspects to these definitions for measuring the output quality. First of all, these definitions are well known from the literature on Information Retrieval (e.g., [Salton and McGill, 1983]) and have proven to be useful, informative, and intuitive measures in many studies in that field and elsewhere (see for instance

[i]Of course, even for KBSs which return a single answer, this assumption still holds since we can interpret such a single answer x simply as the singleton set $\{x\}$.

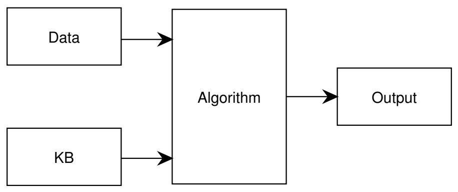


Figure 2.1: Knowledge-Based System components.

[Schumann and Fischer, 1997] for an application of these measures to deduction-based software component retrieval). Secondly, these measures are completely general and make no commitment to either the task or the domain of the KBS that we wish to study. Consequently, the approach proposed can be directly applied to other KBSs, even when they are very different from the one that we happen to have chosen in our own case study.

The above definitions are in fact gradual versions of the classical notions of soundness and completeness: recall corresponds to the degree of completeness of the system, and precision corresponds to the degree of soundness of the system. These measures provide a quantitative angle on earlier work by [vanHarmelen and tenTeije, 1998] which was strictly qualitative.

2.1.3 Knowledge-base system input

We can represent a KBS graphically as in Figure 2.1.3: the data, the knowledge base, the algorithm and the output. By 'KBS input' we mean the data given for example by a user as well as the knowledge base used by the system. The degradation experiments we propose for measuring the robustness of the system will either be with respect to the quality of the data inputted or with respect to the quality of the knowledge base.

For the input quality measure we give two general aspects to consider for measurement. These aspects are incompleteness or incorrectness of the data or knowledge. Although these aspects are quite general, they may not be applicable to every knowledge base. The aspects proposed also do not cover all possible aspects one might want to measure. More explicitly formulated, with incompleteness and incorrectness we mean the following:

Incomplete: A part of the data inputted or part of the knowledge used is missing from the KBS input. For example, the data inputted could be a number of observations. Some of the observations might be to hard or to expensive to obtain. A part of the knowledge might be missing because the knowledge was obtained from human experts who forgot to provide it.

Incorrect: In contrast with incomplete data or knowledge, a certain part of the data or knowledge *is* provided by a user or expert. However, it is incorrectly represented. For example, incorrect data could be caused by a user who has made a wrong observation.

Incorrect knowledge could for example be caused by faulty knowledge of an expert, misunderstandings when coding the knowledge into the system, or limitations of the representation used by the system.

These two aspects are important in KBS validation. For example, a knowledge base will most likely be incomplete and often partially incorrect. Hence, to analyse the robustness of a knowledge base with respect to incompleteness or incorrectness is a realistic and important issue.

Notice that we have already taken a step further than [vanHarmelen and tenTeije, 1998]. In that paper, the authors did not commit to any definition of quality on input or output, and only demanded that whatever the definition was, it should respect a partial ordering. In our approach with degradation testing, we commit to a specific definition of output quality, while leaving input quality open to be defined for each specific application.

2.1.4 Comparing robustness

The only notion that is still left undefined is some ordering on robustness: when do we call a system more robust or less robust than another? Unlike output quality (where we have given a single widely applicable definition) and input quality (whose definition is deliberately left open to depend on the task-type), we have not been able to determine a good answer to this question.

When input quality decreases, a system that produces an output with a fluctuating quality is much less predictable than a system that produces an output with a monotonically decreasing quality. We therefore demand that any system which is called robust at least produces an output with monotonically decreasing quality as function of decreasing input quality.

Definition 2.1.4.1 (Monotonicity) *A robust system will show a monotonically decreasing output quality as a function of deteriorating input quality.*

Note that this demand corresponds precisely to the usual demand on anytime algorithms that their output quality monotonically increases with increasing run-time [Dean and Boddy, 1988]. However, this demand is insufficient for ordering various systems according to their robustness. We therefore give additional competing definitions without choosing one over the other.

Definition 2.1.4.2 (Quality Value) *A system S_1 is more robust than a system S_2 for a set of inputs, if everywhere on that input set the output quality of S_1 is higher than the output quality of S_2.*

Definition 2.1.4.3 (Rate of Quality Change) *A system S_1 is more robust than a system S_2 for a set of inputs, if everywhere on that input set the output quality of S_1 decreases more slowly than the output quality of S_2.*

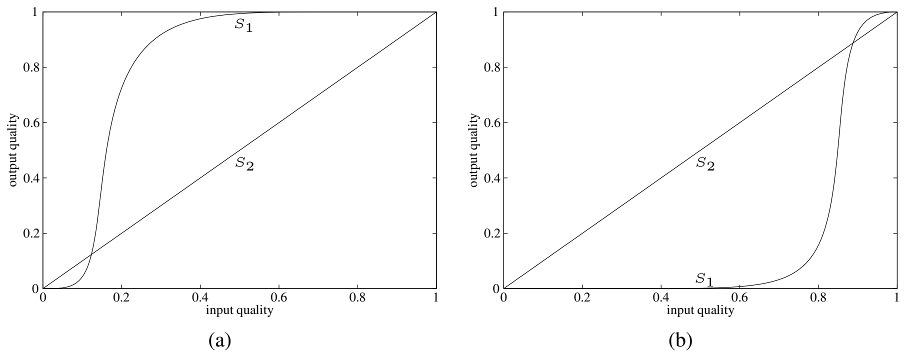


Figure 2.2: Comparing robustness of systems.

Definition 2.1.4.4 (Integral of Quality Value) *A system S_1 is more robust than a system S_2 for a set of inputs, if on that input set the integral of the output quality of S_1 is larger than the same integral for S_2.*

Formally speaking, Definition 2.1.4.2 compares the output quality of two systems (and is concerned with which system produces the *best* output), while Definition 2.1.4.3 compares the first derivative of the output quality of the systems (and is therefore concerned with which system produces the *most stable* output). Definition 2.1.4.4 compares the overall quality of the output quality over an entire interval even when neither system always dominates the other (as required in Definition 2.1.4.2). These definitions are illustrated in Figure 2.2[ii]

Note that the definitions do not have to be applied on the entire input quality range. Instead, they should be applied to an interval of interest. Usually this will be some interval including 1 (i.e., $[x, 1]$).

In Figure 2.2(a), system S_1 is more robust on for example the interval [0.1, 0.3] according to Definition 2.1.4.2, since on that interval its output quality is always higher than that of S_2. However, according to Definition 2.1.4.3, S_2 is the more robust of the two, since its output quality decreases more gradually (reading the graph from right to left). Definition 2.1.4.4 allows to take a more overall perspective: it takes the size of the area under the output quality graph as measure of the overall quality. Under this definition, S_1 is more robust on the entire interval [0,1], since the value of $\int_0^1 output\ quality\ d(input\ quality)$ is larger for S_1 than for S_2. Note that the situation is rather different in Figure 2.2(b). Although the output quality again increases monotonically from 0 to 1 over the same interval, the comparisons between S_1 and S_2 on various subintervals are very different. Now, for example on the interval [0.1, 0.3], S_2 is more robust than S_1 when using Definition 2.1.4.2, but S_1 is more robust than S_2 when using Definition 2.1.4.3.

[ii] In our figures, we plot input quality against output quality. When speaking about 'robustness', we are interested in *decreasing* input quality, so the graphs must be read from right to left.

At the current point in our research, we simply propose each of these definitions as reasonable, without claiming superiority of any definition in all cases. In fact, we believe that under different pragmatic circumstances, different definitions will be preferable: if steep drops in system performance are to be avoided, the second definition is preferable. If one is interested in upholding output quality as long as possible in the face of declining input, the other two definitions may be preferred.

Chapter 3

The degradation studies

In Chapter 2 a methodology for measuring the robustness of KBSs with respect to some input quality is desribed. In this chapter the robustness results are reported on a case study using the proposed methodology.

The structure of this chaper is as follows. We start in Section 3.1 by describing the KBS used in the case study. Then some preliminary notes about the case study are given in Section 3.2. The rest of this chapter reports the robustness results of the case study using the proposed methodology. In Section 3.3 the robustness is analysed with respect to the data input, while in Section 3.4 the robustness is analysed with respect to the knowledge base.

3.1 Example knowledge-based system

For our case study we have used a classification system for commonly occurring vegetation in Southern Germany. The plant-classification system was created with the D3 Shell-Kit which is a tool for the development of KBSs. We will not discuss this tool here but refer to a number of publications about D3 [Puppe *et al.*, 1996, Puppe *et al.*, 1994]. It is also possible to download a demo-version of the software from the URL `http://d3.informatik.uni-wuerzburg.de`.

The plant-classification system that we studied can have 40 different observables as input and has 93 different plant names as output. The knowledge base consists of 7586 rules. Furthermore, with the system we received 150 test cases. Each of these cases consisted of the set of observations for that case (colour and shape of flowers, leafs, stem, etc.), together with the (supposedly correct) answer for these observations as given by a human expert. Around 97% could be answered correctly by the system.

The input observations can be entered in a graphical user interface, but the user is not restricted to the ordering in this interface. The observations can be entered in any order, thus the input can be seen as a *set*. This is not entirely true because some observations are dependent on other observations and will only appear when certain input-conditions are met. Because of these dependencies, the maximum number of observations that can be given for one case is 30.

For our degradation tests we translated the plant classification system into Prolog code. This resulted in a knowledge base with 11724 rules all with the following representation:

```
kb(Plant, Observation, Value, Score).
```

Each time a new observation is given to the system, all rules are collected containing the same `Observation` and `Value`. For each of these rules the score of the `Plant` mentioned in the rule is adjusted by adding the `Score` to its current score. When the score crosses a threshold it is outputted by the system.

In fact the rules do not actually contain numerical scores but descriptions, which were used to make it easier for the experts to express their knowledge. The descriptions are however translated to numerical scores whenever they are used by the system. We will therefore use the descriptions as if they are numbers. The descriptions range from P_1 through P_6 for positive scores and N_1 through N_6 for the negative scores. Furthermore, $|Pi| = |Ni|$ for $i = 1, \ldots, 6$ and $P_{i+1} = 2 * P_i$ for $i = 1, \ldots, 5$.

3.2 Results of the degradation study – preliminary notes

Before discussing the results of the case study in more detail we emphasize that the case study only serves to illustrate the proposal to anlyse the robustness of KBSs through degradation studies. The important aspects of this case study are the quantities measured and how they are analysed, not the obtained robustness results of the plant classification system.

Before we discuss the results, a final remark must be made about the possible values of recall and precision in this case study. Since for every case there is at most one correct answer (namely the name of the actual plant on which the observations were made), we have for any case I, $|correct(I)| = 1$ iff the case is in the knowledge base or $|correct(I)| = 0$ iff the case is not in the knowledge base. As a result, the only values that $recall(I)$ can assume are either 0 or 1. For the same reason, $precision(I)$ is either 0 or $1/|output(I)|$. However, we are not interested in the specific behaviour of the system for a particular case, but in the average behaviour of the system. Hence, we are interested in the average recall (i.e., the sum of all recall values divided by the number of cases used) and the average precision. We will represent these averages in our figures, but will use for example the term 'recall' when in fact we mean 'average recall'.

3.3 Results of the degradation study – data input

In this section we present the robustness results with respect to the data inputted that we have obtained in empirical experiments with the plant-classification system. First, we will define the input quality measure we will use in Section 3.3.1. Thereafter we will give results using the ordering on the observables found in the test cases (Section 3.3.2). We also anlyze the effect of other orderings in Section 3.3.4.

3.3.1 Which input quality measure to use?

According to the definitions from Section 2.1 we must still decide on what to use as a measure on the input quality. In this case study we choose the *completeness of the input* as the measure of input quality. In our classification system, completeness of the input can be directly translated as the *number of available observations*.

There are two reasons why this choice is reasonable and attractive:

Robustness: in many practical classification settings, the input observations are not completely available. It then becomes an interesting question how robust the system functions under such incomplete input.

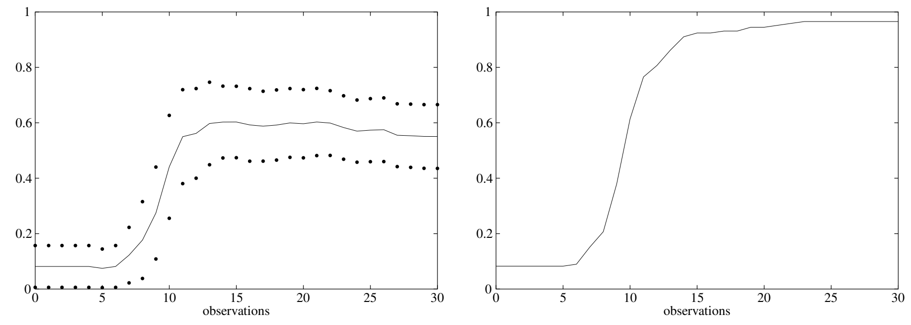
Anytime behaviour: Even when all observations are present, there are practical settings where insufficient run-time is available to process all the observations: some output from the system is required before a given real-time deadline, and not all observations can be processed before this deadline. Those observations that could not be processed before the deadline can be regarded as 'missing from the input'.

As a result of this second reason, the degradation results that we present in this section can also be seen as *anytime performance profiles* for the plant-classification system. Performance profiles are a basic tool in the study of anytime algorithms [Dean and Boddy, 1988]. They plot the output quality as a function of available run-time. Since available run-time can be interpreted as one aspect of 'input quality', such performance profiles are simply a special case of our more general proposal: performance profiles only study output degradation as a function of decreased run-time, whereas our approach is applicable to any aspect of input quality that one chooses to model.

In the following we will present a number of graphs analyzing the robustness of the plant-classification system. Each of these graphs plot output-quality (measured by either recall or precision) against input-quality (measured by the number of observations that were available to the system). If one is interested in anytime behaviour, these graphs can be read from left to right: 'what happens when the system has time to process more and more of the inputs?'. If one is interested in robustness, these graphs should be read from right to left: 'what happens when the system is provided with fewer and fewer of the inputs?'

3.3.2 Using the input sequence from the test-cases

Now that we have established that the number of available observations will be the input-aspect that we will degrade in our studies, we have to decide in which order observations will be made available to the system. Our first choice is simply based on the order in which the observations appeared in the test-cases for the plant-classification system. Each such test-case consisted of a list of observables and their values for that case. Figure 3.1(a) shows how the precision of the answers from the system (as defined in Definition 2.1.2.3) increases when longer initial-sequences of the test-cases were given to the system. The first surprise that this graph has in store for us is its monotonic growth:



(a) Precision (b) Recall

Figure 3.1: Measures with test-case order.

Surprise 3.3.2.1 *Both average precision and average recall (see Figure 3.1(b)) grow monotonically (or: almost monotonically in the case of precision) when adding more observations. This is somewhat surprising since this is not true for individual cases.*[i]

The classification algorithm of the plant-classification system assigns both positive and negative scores. This means that the answer-set can both grow and shrink when adding more observations. In fact, only 58% of the test-cases has a monotonically growing answer-set. As mentioned above, such monotonic behaviour is desirable from both a robustness and from an anytime perspective, so on a case-by-case basis, the plant-classification system does not score very well on this. Surprisingly, the average-case behaviour of the system is apparently much better.

A second observation to make is that (as to be expected) initially the system is not able to make any sensible guess at likely solutions (this holds up to about 6 observations). For higher number of available observations, the graph is surprising for two reasons:

Surprise 3.3.2.2 *After about 12 observations, adding more observations does not increase the precision. This is surprising since most cases contain as much as 19-30 observations. Figure 3.1(a) suggests that 12 observations is sufficient to obtain the maximally achievable precision on average.*

Surprise 3.3.2.3 *The region in which additional observations actually contribute to an increase in precision is surprisingly small, namely between the 6 and 12 observations. Of the 19-30 observations per case, all observable changes to the output seem to be in this small segment of observations!*

Figure 3.1(b) shows similar results for the other dimension of output quality, namely the recall from definition 2.1.2.2.

[i]This result is not completely surprising because it is well known that the average of several variables can indeed show a different distribution than the individual variables.

The dotted lines in Figure 3.1(a) indicate the variance of the precision, and this variance is rather significant. It shows that the distribution of the actual precision-values that were obtained for the different cases are actually spread rather widely around the average.[ii]

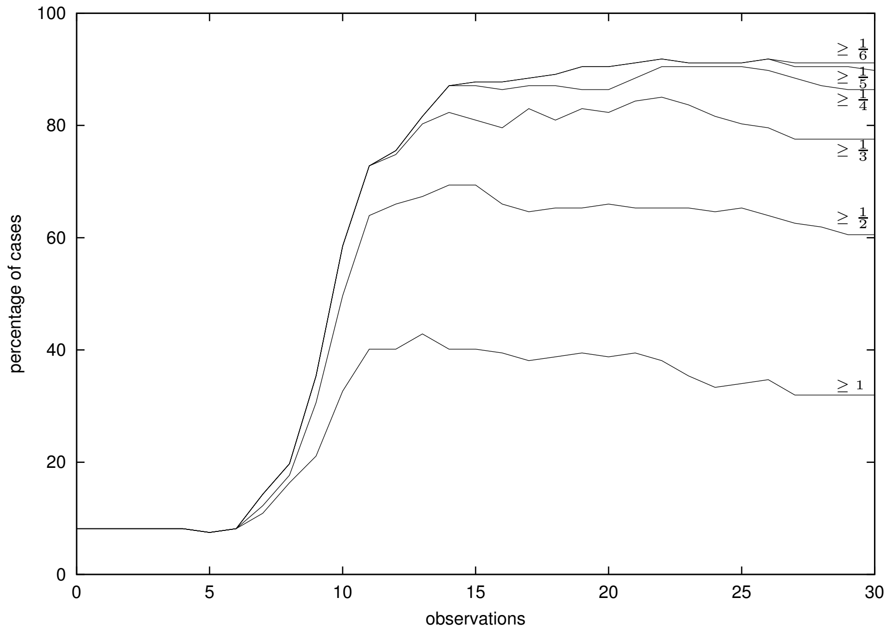


Figure 3.2: Precision with multiple levels.

Figure 3.2 gives more insight in the distribution of the precision than the simple average from figure 3.1(a). Each line in this figure shows the percentage of cases that achieved a precision of at least a certain value after the given number of observations. The lowest line shows that after 12 observations, 40% of the cases have already reached the maximum precision (namely 1). Furthermore, and more surprisingly, this percentage then stops growing! This means that:

Surprise 3.3.2.4 *When aiming for the maximum precision of 1, there is no need to use any more than 12 observations (out of a maximum of 30!). If the maximum precision has not been reached after the first 12 observations, adding further observations will not help.*

This is actually a more precise version of surprise 3.3.2.2 above. There we claimed that extending beyond 12 observations was not useful *on average*. Here we see that for harder cases, a few more observations do actually help, although not more than 20 observations in total.

[ii]No variance was plotted for the recall since, as explained above, in our application the recall is either 0 or 1. Because of this, the variance for recall is not a meaningful notion.

This is because at the other end of the scale (the top line in figure 3.2) , we see that the percentage of cases with a precision of at least 0.2 continues to increase during a longer interval. Apparently, harder cases (those that ultimately achieve a lower precision) benefit more from additional observations than easy cases (those that achieve precision 1). Nevertheless, even there we see that no increase is gained after about 20 observations:

Surprise 3.3.2.5 *Whatever the final precision that is ultimately obtained by the system, this level of precision is already obtained after at most 20 observations. It seems that asking for any more then 20 observations will not improve the output quality any further. This is surprising since many cases (in fact 98% of the test set) contain more than 20 observations.*

Looking at the initial segment of observations, we see another surprise: although we may expect that a low number of observations leads to a low average precision, it is surprising that the lines for the different precision-levels all *coincide* until the 6th observation:

Surprise 3.3.2.6 *No increase in precision can be gained from the first 6 observations.*

This means that in an anytime setting, interrupting the system before the 6th observation is completely useless, since no increase in precision will have been obtained yet.

Figure 3.2 is particularly interesting from an anytime perspective: it tells us for each partially processed input what the chance is that the system has already obtained a certain precision in its output: for instance, after having fed the system 10 observations, there is a 30% chance that it has already obtained the maximum precision of 1, a 45% chance that it has already obtained a precision of at least 0.5, and a 60% chance that it has already obtained a precision of at least 0.3.[iii] This information can be used by the user to determine if it is useful to continue feeding the system more input, or if a sufficiently high precision has already been obtained for the purposes of the user, so that processing (and acquiring potentially expensive observations) can be stopped. Our graph (when interpreted as a performance profile) contains much more information than the usual performance profiles presented in the literature (e.g., [Zilberstein, 1996]). These graphs typically give only a *single* expected value for the output quality at any point in time (compare our Figure 3.1(a)), whereas we give a probability distribution of the expected output value, which is much more informative.

Note that the ideas behind Figure 3.2 can in principle also be applied to the recall. We omitted this because in our case study the recall is either 0 or 1, thus the resulting figure would be the same as Figure 3.1(b).

3.3.3 Early conclusions on robustness

Since we are only studying a single system, we cannot apply the definitions from Section 2.1.4, which only speak about one system being more or less robust than another. However, what is clear from the analysis until now is that the robustness of the plant-classification

[iii]Note that these chances are cumulative, which is why they add up to more than 100%.

system is certainly not very uniform across the distribution of input quality. While degrading input quality (i.e., reading the previous graphs from right to left), the system at first appears extremely stable against missing observations: no quality loss occurs at all. This holds until we are left with somewhere between 12-15 observations (depending on the difficulty of the case). At that point, the robustness of the system is very low, and the input quality drops dramatically.

Is this desirable behaviour or not? Would a more uniform behaviour (e.g., a straight line connects bottom-left and top-right of figure 3.1(a)) be more attractive? This question cannot be answered in general, but depends on the pragmatics of the system in use. The 'straight line' profile is on the one hand more attractive, because it avoids the dramatic drop in quality seen in the figures above (Definition 2.1.4.3 from section 2.1.4); on the other hand, it would start loosing output quality straight away, while the profiles discussed above are all remarkably resistant to quality loss during early phases of input degradation (Definition 2.1.4.2 from section 2.1.4).

3.3.4 Exploring other input sequences

In all the profiles above, we have degraded the input by removing observations in the order in which they were listed in each test-case. Figure 3.3 shows what happens if the input is degraded by removing observations in a different order.

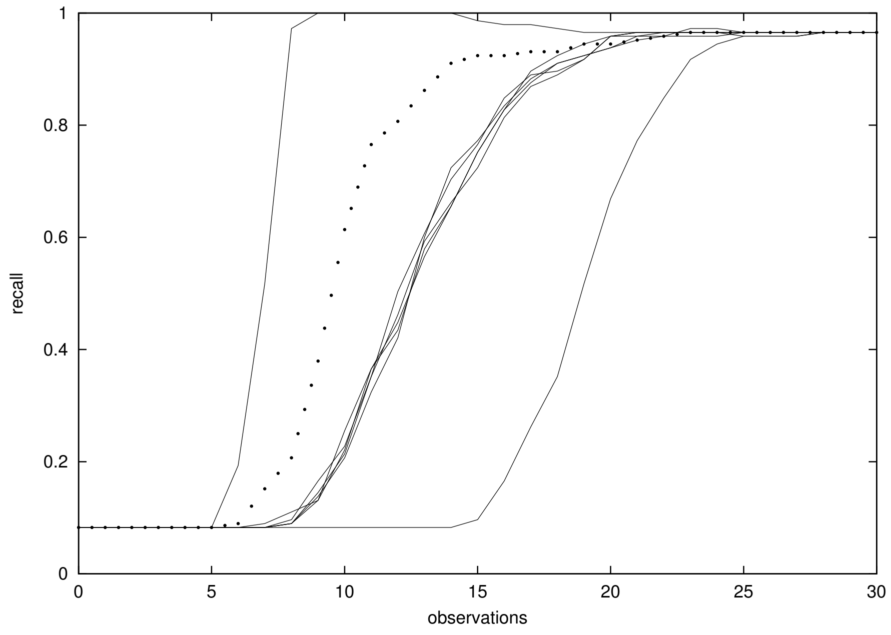


Figure 3.3: Recall with various orderings.

For reference, the dotted line shows the recall-profile from figure 3.1(b). The left-most line shows the theoretically optimal average-recall profile: at each step in each case, we computed which next observation would contribute maximally to an increase in recall. Of course, this cannot be done in practice, since which observation will contribute most in general depends on the observation-value that is obtained, so this computation can only be done theoretically for test-cases where all observations are already present. The value of the left-most line is therefore only to show what would be the theoretically fastest increase in recall by the system with the fewest possible observations. The right-most line in Figure 3.3 does the same, but this time for the theoretically slowest average-recall profile. Every other possible recall profile must lie between these two lines (as is indeed the case with the earlier observed profile based on the test-case sequence). Finally, Figure 3.3 shows a narrow bundle of recall profiles. Each of these profiles corresponds to a randomly generated order of the observations.

The dotted line in this figure shows the profile based on the observation order obtained from the test-cases (as originally plotted in Figure 3.1(b)). We can now see that this observation order actually scores rather well when compared with the random sequences:

Surprise 3.3.4.1 *The degradation sequence taken from the test-cases is surprisingly effective in obtaining a high recall after only a few observations. In fact, it is much closer to the theoretically optimal sequence than the randomly generated (information-free) sequences.*

Currently, we have no good explanation for this phenomenon. It is possible that the order of the input observables in the test-cases is influenced by the order in the graphical user-interface. We will look more closely at this ordering in Section 3.3.6.

3.3.5 Further conclusions on robustness

The variation in the curves from Figure 3.3 shows that the plant-classification is very sensitive to the specific order in which the observations are presented to the system. In other words: when the same set of observations are presented to the system in a different order, the behaviour of the system may change dramatically.

This type of robustness is not covered by our robustness definitions in Section 2.1.4. The definitions there are all concerned with comparing the behaviour of different systems on the same degrading input. The phenomenon observed in Figure 3.3 concerns the behaviour of a single system on different ways of degrading the input. We leave it for further research how to include this type of robustness in our approach.

3.3.6 Which sequence does the system actually use?

The original plant-classification system has been designed in such a way that all observations can be entered at any time during the dialogue. The system does not enforce a particular order among the observations to be entered as input. Nevertheless, the user-interface

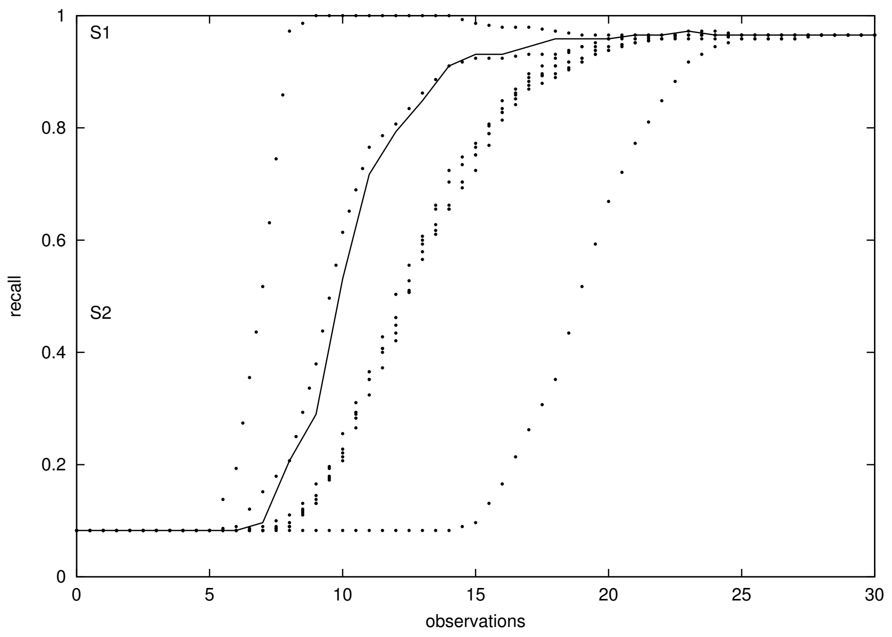


Figure 3.4: Recall with user-interface ordering.

of the system does suggest a particular input sequence, namely the order in which the observations occur on the input form. Although the user can enter observations anywhere on the input form, the top-to-bottom sequence of this form is suggestive.

The solid line in Figure 3.4 shows the increase in recall for this user-interface sequence. The dotted lines show the theoretically fastest, the theoretically slowest gains in recall, and the gains from random observation orderings and the test-case ordering as well (all copied from Figure 3.3). As can be observed in this figure, the user-interface sequence performs rather well when interpreted as an anytime performance profile (and certainly much better than a random ordering). This suggests that the user-interface has to some extent been designed with this behaviour in mind. Although this behaviour was not the reason for the plant-classification system, it would be interesting to see how a dedicated dialogue strategy can save time.

Actually, the real reason behind the ordering of the user-interface was the intended use of the system. This means that we have to place our results in this context. The ordering of the user-interface is based on the way people describe plants. First the system asks about the flower, which is the most specific part of the plant and asks about the leaves and stem afterwards. This specificity ordering may be the reason for the better curve in figure 3.4 when we compare it with a random ordering.

Furthermore, we like to note that the most optimal input order with respect to the recall or precision, may not be realizable in practice because it can decrease other factors like for example the user-friendliness of the user-interface. Some trade off will usually have to be made.

3.4 Results of the degradation study – knowledge base

The degradation experiments on data input already yielded interesting insights in the behaviour of a realistic KBS. In this section we will perform the same kind of analysis with respect to the knowledge base used. For the input quality we will measure the incompleteness and incorrectness of the knowledge base. In Section 3.4.1 we will analyse robustness with respect to an incomplete knowledge base, whereas in Section 3.4.2 we will analyse robustness with respect to an incorrect knowledge base.

3.4.1 Incomplete knowledge

Our experiment is as follows: we select some percentage of rules at random from the knowledge base, remove the selected rules, and compute the recall and precision value for each case with the modified knowledge base. The average recall (i.e., the sum of all recall values divided by the number of cases) and the average precision will indicate the robustness of the system with respect to the percentage of rules removed. Of course, the experiment has to be repeated multiple times to obtain the robustness of the system on average. Note that as we are no experts ourselves in plant classification and therefore unable to check if some rules are missing from the knowledge base, we will call the provided plant classification system complete, although in reality it may not be.

The results of the experiments are reported in Figure 3.5(b) for the recall and in Figure 3.5(a) for the precision. For each percentage, the average recall and average precision was obtained over 10 different runs. Besides average values, the minimal and maximal obtained values are also plotted in both graphs.

Both graphs show an almost smooth decrease of our measured values. The graphs are surprising in that the measures start to decrease around 40% in both graphs.

Surprise 3.4.1.1 *In the plant classification system almost 40% of the knowledge base can be removed at random without severe quality loss.*

A possible cause for this decrease at 40% and not at some earlier point is that the average recall and precision are computed over 150 cases. When a rule is removed from the knowledge base it will only have effect on a small number of cases and therefore only a small effect on the average recall and precision. When more rules are removed, more cases will be effected which results in a decreasing recall and precision.

This form of degradation experiment can lead to interesting insights into the robustness behaviour of the system analysed. Nevertheless, the experiment performed has a drawback. It is not realistic to assume that each element in the knowledge base has the same chance of being forgotten. It would be more realistic to remove parts of the knowledge base based on some probability measure. Several measures should be compared when it is not obvious which measure should be used.

In another experiment we partitioned the knowledge base into 7 partitions. The partitioning was based on the scores given by the rules. Two rules were put into the same class whenever their scores were the same or only differed in their sign (e.g., a rule with a score of 20 would be put into the same class as a rule with a score of 20 or -20). (We will refer to each class by a number and the smallest numbered class contains the rules with the smallest positive scores.) We then defined a probability measure over this partition (Figure 3.6).

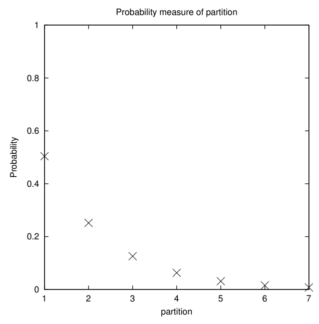


Figure 3.6: Probability measure.

The reasoning behind our choice for the chosen probability measure was the following. The partitioning is based on the size of the scores of the rules. Rules in a smaller numbered class have less effect on the final outcome than rules in a higher numbered class. It seemes that in this domain rules with a smaller effect might easier be forgotton. The probability measure in Figure 3.6 was chosen in such a way that when the score of a rule A is twice the size of the score of a rule B then the probability to choose A would be halve the probability of rule B.

Our results with this probability measure are shown in Figure 3.7. Obviously the results have changed dramatically. Whereas our previous results (Figure 3.5) showed that 40% of the knowledge base could be removed without severe quality loss, the new results show

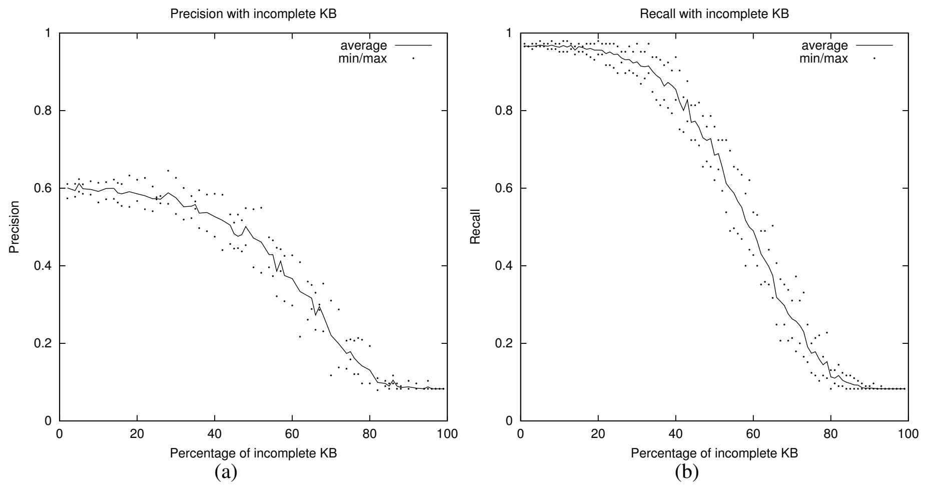


Figure 3.5: Recall and precision with incomplete knowledge base.

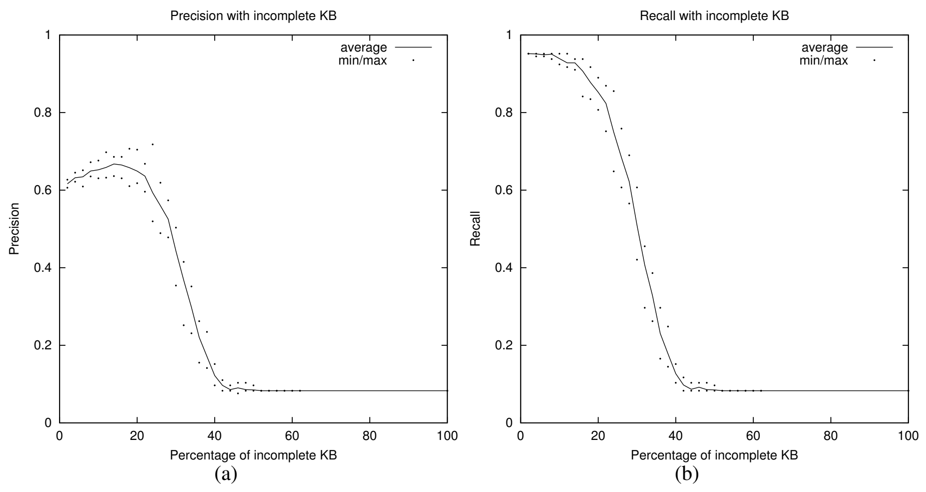


Figure 3.7: Recall and precision with incomplete knowledge base using a probability measure.

that when 40% of the knowledge base is removed in a biased way the system becomes useless. Furthermore, the drop in output quality already starts at 15% and is much steeper than before.

These new results should warn people before using the degradation studies for analyzing the robustness of KBSs. The experimental setup should be as realistic as possible and care has to be taken when interpreting the results as these can be sensitive to the chosen setup as we demonstrated with our case study.

3.4.2 Incorrect knowledge

In our case study, the knowledge base consists of rules. Each rule consists of some observations, a name of a plant, and a score. To test the robustness with respect to incorrect knowledge the rules need to be modified in some way. However, we like to do this in a realistic setting. Modifying the observations at random is therefore no option. First of all some observations are usually answered right while others are prone to errors. Secondly, if an observation is wrong it is usually not done at random. Some answers lookalike while others are clearly distinct. Modifying the observations of the rules realistically therefore requires domain knowledge which we lack. However, modifying the score of a rule in our case study does not require any domain knowledge.

In our experiments with incorrect knowledge we address the question: "How robust is the system for incorrectly entered scores?". Our experiment is therefore as follows: we select some percentage of rules at random from the knowledge base, modify the score of the selected rules, and compute the recall and precision value for each case with the modified knowledge base. The average recall (i.e., the sum of all recall values divided by the number of cases) and the average precision will indicate the robustness of the system with respect to the percentage of rules modified. Of course, the experiment has to be repeated multiple times to obtain the robustness of the system on average.

Only one question remains: “In what way do we modify the score of a rule?” We identified the following parameters:

Direction: Scores can be changed to a higher or lower value. Most likely scores are damaged in both directions. Nevertheless, we also investigate the effect of a biased expert, i.e., an expert who always assesses something too high (or too low). We will use the word ‘positive’ when scores are only changed to a higher value, and the word ‘negative’ when scores are only changed to a lower value. When both words do not occur, scores will be changed in both directions.

Size: The scores of rules are changed from one class to another. Most likely the new class is only one higher or one lower than the old class. We will call this kind of damage a ‘near miss’. To be complete we also investigate the effect when the difference of the new class and old class is two classes. We will call this kind of damage an ‘error’.

The parameters lead to six different experiments, but before discussing the actual results let us first consider what results are to be expected.

3.4.2.1 Hypotheses

The plant classification system we are analyzing uses a threshold and only gives plants with a score higher than this threshold as output. When we change the scores of rules to a higher value it follows that the final score of the plants can only increase. As the recall only measures if the correct answer is given as output it follows that the recall can never decrease. The recall will probably increase as more plants (including the correct answer) are now more likely to cross the threshold and be given as output.

Hypothesis 3.4.2.1 *When the scores of rules are increased, the recall of the system will increase.*

The converse also holds. When we change the scores of the rules to a lower value it follows that the recall can never increase. When the scores are decreased sufficiently the recall will decrease as outputted plants which are correct will drop below the threshold.

Hypothesis 3.4.2.2 *When the scores of rules are decreased, the recall of the system will decrease.*

Some hypotheses for the precision can also be given, but the argument is more complex. Considering the formula for the precision (Definition 2.1.2.3) we can identify two causes for an increase in precision:

I1. The score of the correct plant is increased and thereby crosses the threshold. In this case the precision changes from 0 to some positive value.
I2. The scores of one or more incorrect plants decrease and thereby drop below the threshold. If the correct answer remains in the output this will increase the precision from some positive value to a higher positive value. For example, let A and B be two plants given as output. Let A be the correct plant. In this case the system will have a precision of $1/2$. When the scores are changed such that B drops below the threshold while A remains above the threshold the precision will change to 1.

In a similar way we can identify two causes for a decrease in precision:

D1. The score of the correct plant drops below the threshold. The precision changes from a positive value to 0.
D2. The score of one or more incorrect plants increase and cross the threshold.

When we change the scores of rules to a higher value both I1 or D2 can occur. However, I1 is unlikely as previous experiments showed that the recall of the system is already close to the optimal value of 1. It follows that we expect a decrease in precision.

Hypothesis 3.4.2.3 *When the scores of rules are increased, the precision of the system will decrease.*

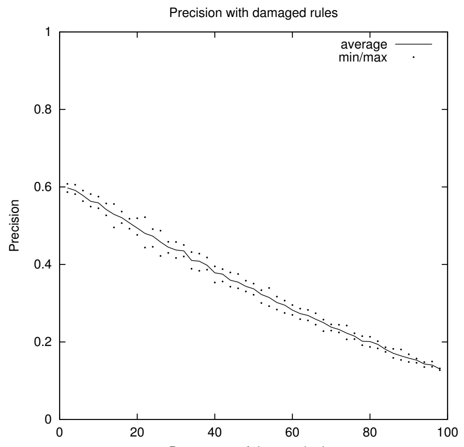
When we change the scores of rules to a lower value both I2 or D1 can occur. As these causes change the precision into two different directions it is not obvious to predict the outcome of the precision.

The outcome is even harder to predict when we change the scores of rules both to lower values and to higher values as all four causes for the change in precision can occur. Also the recall can now both increase and decrease and is therefore harder to predict. Nevertheless, for the recall we know that the performance profile has to lie somewhere in between the results for the strictly positive and negative changes.

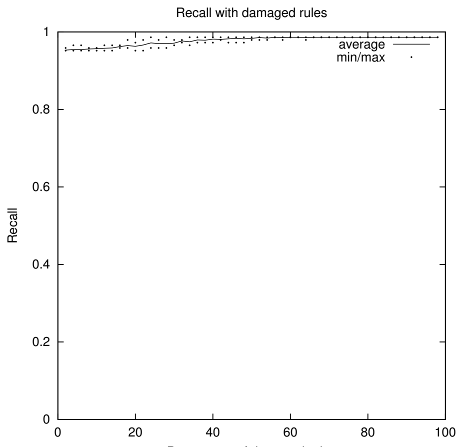
Hypothesis 3.4.2.4 *When some rules are changed positively while other rules are changed negatively, the recall of the system will be between the recall values of the stricly positive and negative changes.*

Besides direction of changes we will also look into the effect of the size of the changes. We expect that a larger chance will have a larger effect on the average values. For example, when the scores are changed positively and the recall increases with near misses, we expect the recall to increase even more when we experiment with positive errors.

Hypothesis 3.4.2.5 *When comparing experiments in which we make errors instead of near misses without changing the direction of the changes, the effect observed in the near miss experiment will be made stronger.*

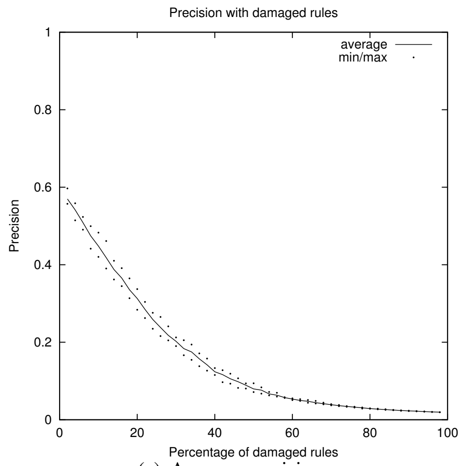


(a) Average precision

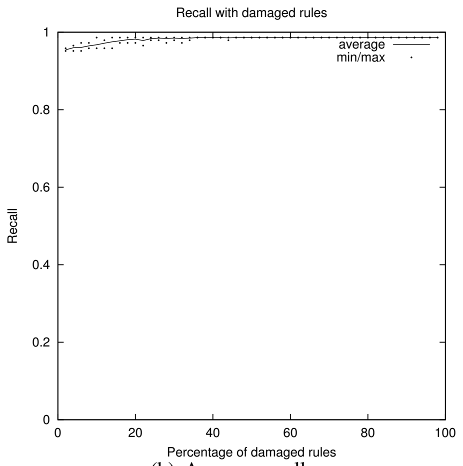


(b) Average recall

Figure 3.8: Positive near misses.



(a) Average precision



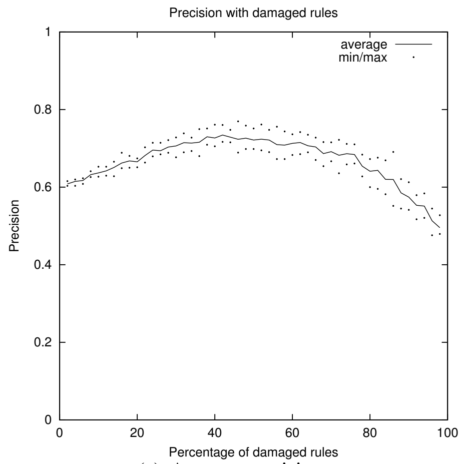
(b) Average recall

Figure 3.9: Positive errors.

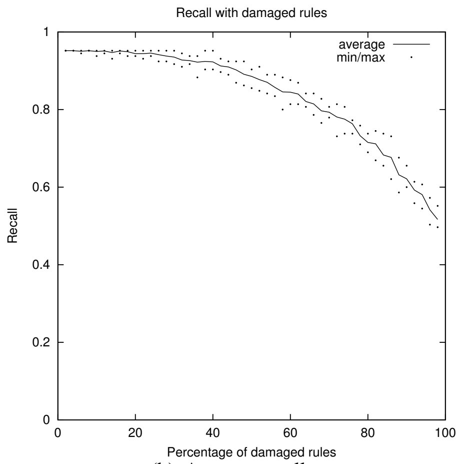
3.4.2.2 Results

Now we will discuss the results which are shown in the Figures 3.8 through 3.13.

First note that our hypotheses hold in the shown figures. In case of positive changes the recall increases as is shown by Figures 3.8(b) and 3.9(b) (Hypothesis 3.4.2.1). In case of negative changes the recall decreases as is shown by Figures 3.10(b) and 3.11(b) (Hypothesis 3.4.2.2). In case of positive changes the precision decreases as is shown by Figures

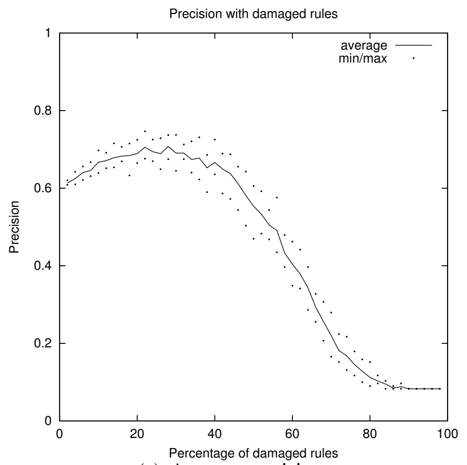


(a) Average precision

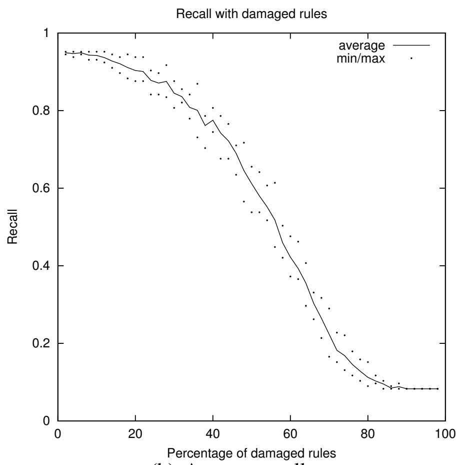


(b) Average recall

Figure 3.10: Negative near misses.



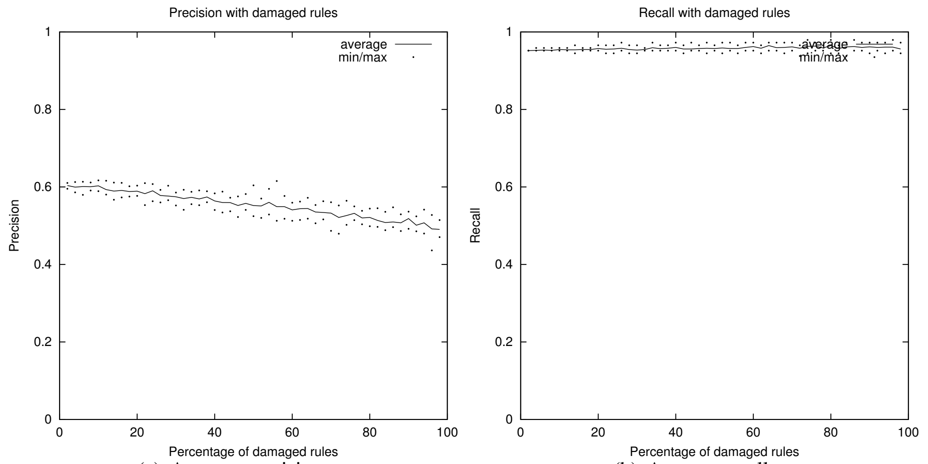
(a) Average precision



(b) Average recall

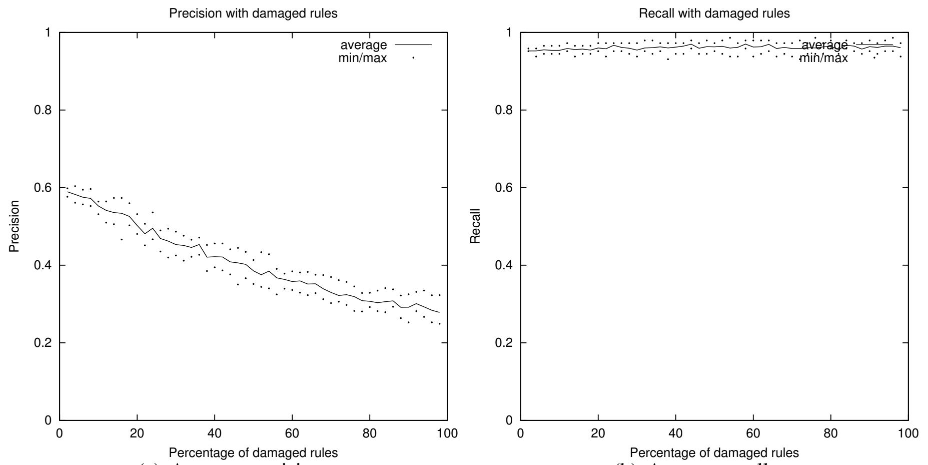
Figure 3.11: Negative Errors.

3.8(a) and 3.9(a) (Hypothesis 3.4.2.3). In case of near misses, the recall (Figure 3.12(b)) will lie between the values found in the strictly positive experiment (Figure 3.8(b)) and the strictly negative experiment (Figure 3.10(b)). The same holds for the experiments with errors (Figures 3.13(a), 3.9(a), and 3.11(a)) (Hypothesis 3.4.2.4). In case of positive changes, we find that the recall in the experiment with errors (Figure 3.9(b)) lies everywhere above the recall in the experiment with near misses (Figure 3.8(b)). In case of negative changes,



(a) Average precision (b) Average recall

Figure 3.12: Near misses.



(a) Average precision (b) Average recall

Figure 3.13: Errors.

we find that the recall in the experiment with errors (Figure 3.11(b)) lies everywhere below the recall in the experiment with near misses (Figure 3.10(b)) (Hypothesis 3.4.2.5).

Other points of interest are the specific curves we obtained in our experiments. For example in case of positive near misses (Figure 3.8(a)) we find a *linear* decrease in precision whereas the decrease is *non-linear* when we increase the size of the changes (Figure 3.9(a)).

In the case of negative near misses we could not predict the behaviour of the precision as there are two causes that could change the precision in opposite directions (causes I2 and D2 discussed in Section 3.4.2.1). Figure 3.10(a) tells us that I2 happens more often than D1 in the first segment of the figure as the precision increases. Hence, in most cases first some incorrect plants drop below the threshold before the correct plant drops below the threshold. It follows that in most cases the correct plant has outscored most of the incorrect plants, which is desired behaviour of the system.

The same curve as in negative near misses also occurs in the figure for negative errors (Figure 3.11), however, the curve of the negative near misses is 'compressed' into the first halve of the figure because the size of the changes has increased. Note that this holds for both the precision and the recall.

Besides interesting behaviour we are of course interested in the robustness of the system. Probably the most realistic setting is the experiment with near misses (Figure 3.12). Both precision and recall show a very stable curve indicating a robust knowledge base for 'near misses'. Although the knowledge base appears to be robust on the entire range of the x-axis, from a practical point of view usually only the first part of the figure is important.

Chapter 4

Extensions and conclusions

In Chapter 2 a methodology is presented for measuring the robustness of a KBS with respect to some input quality. In Chapter 3 an experimental analysis of this methodology is presented on a large plant-classification system. In this final chapter of Part I some extensions to the methodology are discussed in Section 4.1. Thereafter, in Section 4.2 conclusions are given about the results of the experimental analysis. The chapter finishes by looking at which steps can be taken for future research (Section 4.3) and acknowledges the people that contributed to this research (Section 4.4).

4.1 Extensions

Degradation experiments can lead to interesting insights into the bahaviour of a KBS. These insights can be used to compare various systems or to improve a system. For example, the plant classification system we analysed shows a very high recall, but its precision is much lower. However, the degradation experiments we performed clearly shows that the precision of the system can be improved (Figure 3.10).

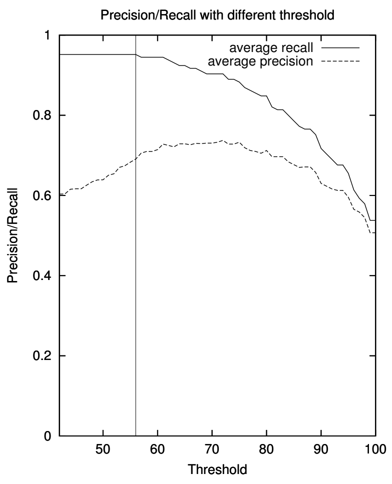


Figure 4.1: Parameter tuning.

The increase in precision in Figure 3.10 can be explained by cause I2 discussed in Section 3.4.2.1. It seems that the score of the correct plant is higher than most plants in the output. Instead of changing the score of rules to a lower value to increase the precision, we can obtain the same effect by setting the threshold used by the system at a higher value. Hence, in this case we use degradation experiments to tune a parameter of the system.

The results of tuning the threshold parameter are shown in Figure 4.1. As can be seen, the precision can be increased to 0.69 without any loss in recall by increasing the threshold to 56. If the threshold is set above 56 some tradeoffs will have to be made. The precision can be improved some more, but the recall will decrease.

4.2 Conclusions

In this part of this dissertation, we have argued for the need for *quantitative analysis* of the quality of KBSs. In particular, we have shown how *robust behaviour* in the light of incomplete system-input as well as an incomplete and incorrect knowledge base is amenable to such quantitative analysis. Our quantitative analysis is based on the idea of *degradation studies*: analyse how the quality of the output changes as a function of degrading input. We have proposed a set of *general definitions* which are general enough that they can be used in similar degradation experiments by others, even if the systems concerned are of a very different nature than the one in our case study. We have shown the practicality of our approach by applying it to a particular *case study*. This yielded a number of surprising insights into the behaviour of the system under study.

We believe that the following issues are the most important in our proposed approach of degradation studies for measuring the robustness of KBSs:

Generality: Degradation studies are general enough to be applicable to many KBSs. The output quality is measured using the recall and precision, which are well known in information retrieval. Furthermore, the general concepts of incompleteness and incorrectness can usually be used as measures for the quality of the input.

Insights: Degradation studies provide insights into the behaviour of the system analysed. As we demonstrated in our case study this includes its anytime performance with respect to incomplete data or its robustness with respect to incomplete or incorrect data or knowledge.

Performability: The degradation study we performed could quite easily be performed. We were able to do this because we had easy and quick access to the KBS (i.e., giving input to the system and reading its output by an external program). Furthermore, programs could be used to modify the input and knowledge base in a controlled way. These conditions should not be too hard to realize for many other KBSs.

Setup: Before performing a degradation study on a KBS, one should carefully consider the experimental setup. The way data or knowledge is removed or modified may greatly influence the outcome of the degradation studies.

The ultimate suggestion that follows from this work is that any KBS should upon delivery come accompanied with a set of degradation statistics such as discussed in this paper as a quantitative way of measuring interesting and important aspects of the systems quality. This would contribute to a more empirical and quantitative analysis of AI systems in general and of KBSs in particular, very much in the spirit of [Cohen, 1995].

4.3 Future work

Although the results above already yield interesting insights in the behaviour of a realistic KBS, many other aspects could still be uncovered using further degradation studies. We discuss some of these extensions in this section:

The informal definition for robustness that we used as a starting point in Section 2.1 has not been carried through entirely in the paper. Functioning correctly in the presence of invalid inputs has not been evaluated in the case study and should be included in future research.

[Zilberstein and Russell, 1995] suggests a category of measures on output quality which is not yet covered by the recall and precision that we have used in the above, namely "specificity of a solution". This is intended to represent the degree of detail in the systems' answer. An example would be a system which can compute names of ever finer grained plant-families instead of only individual species (as above). This property was irrelevant in our case-study, since the plant-classification system only deals with a flat list of candidates, not with a hierarchically organized space of candidates. We have therefore ignored this potential third dimension of KBS output quality, but we expect that good measures can be devised for this just as well as for the other two dimensions which we did handle.

The output quality measures we used (precision and recall) are geared towards systems with a discrete output (a set of answers). Some KBS applications return real-valued answers (e.g. ratings). We must study how these systems can also be subjected to degradation studies using acceptable measures. In fact, the plant-classification system not only returns a set of candidates, but indicates a numeric score for each candidate. Our current output-measures completely ignore this score. A further step would be to also include this score in the quality measures.

As mentioned in Section 3.3.2, Figure 3.2 can be interpreted as a prediction for the expected quality of the output after a given number of observations. In effect, Figure 3.2 is the result of *learning the anytime performance profile* through the test-cases. As with any learning task, we can apply cross-validation to the set of test-cases [Cohen, 1995]: use a subset of the test-cases to "learn" profiles as in Figure 3.2, and use the remaining cases to check the accuracy of the predicted performance levels.

Our measures for output quality (recall and precision) can only be computed for cases where the correct answer is actually known. This is not as obvious as it may sound. In many applications (e.g., computing the best solution to a design problem) the correct (i.e., best) answer is not known to any human expert. In such cases, one must either resort to known approximations of the correct answer, or fundamentally different quality measures must be defined.

Our proposed definitions for output quality (precision and recall) focus on the correctness of the answers computed by a system. Of course, there are many more aspects to the "quality" of a KBS, such as the quality of its explanation, its computational efficiency, its interaction with its environment (be it users or other systems), etc. It is an open issue to us whether the same "degradation study" approach can be taken to quantifying any of these other aspects of the systems quality.

4.4 Acknowledgements

We like to thank Frank Puppe for the help he has given us. This paper was inspired by a discussion with him during the Banff'98 KAW workshop. He gave us access to the plant-classification software and explained in detail its working. He was also always available to answer any question by e-mail. Furthermore, we thank Eric de Vink for his contributions to the early phases of this research and Nico Vink for participating in the experimental analysis of the last phase of this research.

Part II

Knowledge Compilation

Chapter 5

Knowledge compilation

5.1 Introduction

Many systems in symbolic problem solving use logic as representation language. This language can then be used to express basic truths from which other truths can be derived mechanically by logical deduction. The first researcher who proposed the use of logic for problem solving was probably John McCarthy [McCarthy, 1959] and since then various researchers have advocated the favour of the logical approach. We will recall some of those benefits.

First, logic comes with a formal semantics, which gives a precise description of the meaning of expressions in the formalism. With this precise semantics, several logical languages can be compared with each other, which makes it possible to construct guidelines that correlate characteristics of particular domains and tasks with appropriate logic based representational and inferential mechanisms. From this, it is possible to make an informed choice for a representation formalism.

Secondly, logic has well understood properties as regards to soundness, completeness, and decidability. Although some of the results are negative (i.e., semi-decidable, incompleteness), it is important that these results are known. For some other types of representation languages, no results have been obtained.

Thirdly, logic is expressively powerful. There are two reasons for this. First, logic is not restricted to the standard two-valued, truth functional, first-order predicate calculus. Many researchers have already proposed many alternative forms of logic. Examples are modal logic, intuitionistic logic, epistemic logic, and tense logic. The other reason for logic to be expressively powerful is its ability to express what might be called incomplete knowledge. In logic, it is possible to express facts without some details that are not yet known. For example in classical predicate calculus we are able to express the knowledge that an object x exists with property P without knowing its identity: $\exists x : P(x)$. Another example is the disjunctive knowledge $P \vee Q$ without stating, which disjunct is true.

However, the logical approach also has some drawbacks. The complexity of logical entailment is such a drawback. It is well known that deduction in a logical formalism is very much demanding from a computational point of view. Many problems and tasks (e.g., planning, diagnosis, and configuration) that we are typically dealing with in symbolic problem solving are already intractable for the simple varieties. Since such tasks still have to be performed, several methods were developed to deal with this kind of problem.

A technique called *knowledge compilation* is such a method that can be used to deal with computational difficulties. The underlying idea of knowledge compilation is that many reasoning problems can be split into two parts: a knowledge base and a query. For example, in diagnosis the knowledge base consists of rules and facts about the way some system is expected to behave. When there exists a discrepancy between the observed behaviour and the way the system is expected to behave, the knowledge base is queried to give a cause for this discrepancy. In this case the query can be a conjuction of specific facts reflecting the current state (i.e., observations), which implies the cause for discrepancy (in the context of the knowledge base). More specifically, in diagnosis the knowledge of the expected behaviour of a system is represented by a theory T, the current state is represented by some formula F (e.g., a conjunction of facts), and some cause is represented by a literal l. The problem of determining $T \cup F \vdash l$ is logically equivalent to $T \vdash F \Rightarrow l$. This problem can be considered to have two parts: the theory T is the 'knowledge base', and $F \Rightarrow l$ is the 'query' of the problem.

In a typical scenario, the knowledge base remains unchanged over a long period of time and is used to answer many queries. In knowledge compilation the idea is to split this kind of reasoning into two phases:

1. In the first phase the knowledge base is pre-processed by translating it into an appropriate data structure, which allows for more efficient query answering. (This phase is also called *off-line reasoning*.)
2. In the second phase, the data structure, which resulted from the previous phase, is now used to answer the query. (This phase is also called *on-line reasoning*.)

The goal of the pre-processing is to make on-line reasoning computationally easier with respect to reasoning in the case when no pre-processing is done at all.

Pre-processing is quite common in Computer Science. For example, compilers usually optimize object code or a graph can be pre-processed to obtain a data structure that allows for a fast node reachability test. However, in Computer Science pre-processing is usually done for problems, which are already solvable in polynomial time. What characterizes the same study of such techniques in the context of symbolic problem solving is that reasoning problems are often NP-hard.

The rest of this chapter is divided as follows. In Section 5.2 we give the terminology we will use as well as some formal definitions. Thereafter, we discuss several methods used in knowledge compilation. These methods are divided in exact methods (Section 5.3) and approximate methods (Section 5.4).

5.2 Knowledge compilation: terminology

First we introduce a simple reasoning problem which will be used as running example. A reasoning problem is always specified by means of (1) its instances, and (2) the question we are expected to solve. The *Literal Entailment problem*, which is our running example, is specified as follows:

Instance: Finite set L of Boolean variables, a propositional formula in Conjunctive Normal Form T, and a literal l (both T and l are built using variables in L).

Question: Is it true that all models of T are models of l (i.e., that $T \models l$)?

Usually a problem is represented as a pair `Instance/Question`. However, this representation does not tell us which part of `Instance` is fixed. Another representation is therefore needed that clearly splits an instance into a fixed and a variable part. For example, we stated before that many reasoning problems can be considered to consist of two parts: a knowledge base and a query. The knowledge base is not changed often and can therefore be pre-processed and used to answer many problem instances. The query on the other hand is posed to the knowledge base and will be different for each instance. To address the pre-processing ascpects in knowledge compilation, we will therefore use the following terminology [Cadoli, 1993, Cadoli, 1996]:

Fixed part of a problem: The part of a problem that goes to pre-processing when a problem is compiled (e.g., a propositional formula T in CNF). [i]

Variable part of a problem: The part that does not go through pre-processing (e.g., a literal l).

(Structured) problem: A triple consisting of the type of question that we ultimately want to solve, its fixed part, and its variable part (e.g., $[T \models l, T, l]$).

Which part of the problem is considered fixed or variable may depend on the knowledge about the domain. Our experiments, which we will discuss in Chapter 7, confirm that there is a trade-off between the part of the problem that is considered fixed and the effectiveness of the knowledge compilation techniques applied.

We stated before that the goal of knowledge compilation is to make on-line reasoning easier with respect to reasoning in the case when no pre-processing is done at all. An example problem for which this goal can be attained is our running example the Literal Entailment problem. Without pre-processing, the Literal Entailment problem is coNP-complete, but after compiling it, the problem $[T \models l, T, l]$ can be solved on-line in time polynomial in $|T| + |l|$. This can be done as we can record on a table, for each literal

[i]Note that the vague term 'the part' is intentionally used as knowledge compilation can be used on many kinds of data structures (e.g., formulas, models). However, within this thesis we only consider knowledge compilation of propositional formulas and 'the part' can be considered to be a set of propositional formulas representing for example a knowledge base.

l occurring in T, whether $T \models l$ or not. The size of the table is in $O(n)$, where n is the cardinality of the alphabet L of T. The table can be consulted in $O(n)$ time. Note that creating the entire table means solving $O(n)$ instances of a coNP-problem, but this is done off-line and in knowledge compilation one is not concerned with these off-line computational costs.

The compilation of our running example $[T \models l, T, l]$ contains two aspects which are important:

1. The output of the pre-processing part is a data structure (e.g., a set of propositional formulas), which has size polynomial with respect to the fixed part.
2. On-line reasoning takes time polynomial in the size of the data structure and the size of the variable part.

Furthermore, it is believed that the same pre-processing should facilitate the answer to a whole class of queries – not just one. Intuitively, the effort spent in the pre-processing phase pays off when its computation time is amortized over the answers to many queries. Finally, even if the compilation process can take a substantial amount of time, it must always terminate. The aspects mentioned above can be used as guidelines to formalize the notion of a compilable problem. The following definition is from [Cadoli *et al.*, 1994]:

Definition 5.2.0.6 (Compilable problem) *A problem [P, F, V] is compilable if there exist two polynomials p_1, p_2 and an algorithm ASK such that for each instance f of F there is a data structure D_f such that:*

1. $|D_f| \leq p_1(|f|)$.
2. *for each instance v of V the call ASK(D_f, v) returns yes if and only if (f, v) is a "yes" instance of P.*
3. *ASK(D_f, v) requires time $\leq p_2(|v| + |D_f|)$.*

Remember, that in Definition 5.2.0.6 P stands for the type of question we ultimately want to solve. For example, does the knowledge base entail a certain literal, or can a certain cause explain the discrepancy between observations and system description. Furthermore, in Definition 5.2.0.6 F stands for the fixed part of the problem, and V stands for the variable part of the problem. In Definition 5.2.0.6, Constraint 1 states that the size of some data structure D_f is polynomial in the size of some instance f of F. Constraint 2 states that D_f can be used to answer an instance (f, v) of P for any instance v of V. Hence, from Constraints 1 and 2 follows that D_f stands for the compilation of f. Finally, Constraint 3 states that any instance (f, v) can be answered in time polynomial in the size $|v| + |D_f|$.

In case Definition 5.2.0.6 does not hold for a problem $[P, F, V]$ such a problem is said to be *incompilable*.

In the rest of this chapter, we will describe various knowledge compilation methods (for propositional theories). The methods are divided into methods which exactly translate the original theory into another form (Section 5.3) and methods which translate the original theory into a form that approximates the original theory (Section 5.4).

5.3 Exact knowledge compilation

A knowledge compilation method is called exact when the original theory is compiled into a *logically equivalent* theory. Proposals for exact knowledge compilation can be classified in three main methods [Cadoli, 1993]:

1. use prime implicants or prime implicates.
2. add to the knowledge base only those prime implicates that make any deduction possible by unit resolution.
3. use prime implicates with respect to a tractable theory.

Note that an *implicant* of a theory Σ is a conjunction of literals D such that $D \models \Sigma$ and D does not contain two complementary literals; a *prime implicant* is a minimal implicant with respect to set containtment. An *implicate* of a theory Σ is a disjuction of literals C (a *clause*) such that $\Sigma \models C$ and C does not contain two complementary literals; a *prime implicate* is a minimal implicate with respect to set containment.

5.3.1 Prime implicants and prime implicates

The simplest proposals on exact knowledge compilation use the fact that knowledge bases have normal forms (e.g., CNF and DNF) from which consequences can be easily computed.

For example by taking the disjunction of all prime implicants $D_1, \dots, D_k$ of a knowledge base KB one obtains a DNF formula $D_1 \vee \dots \vee D_k$ which is equivalent to KB [Quine, 1959], such that for every query Q, $KB \models Q$ iff for every prime implicant D_i, $D_i \models Q$. If Q is in CNF this amounts to verify that every clause of Q has a non-empty intersection with each D_i. Hence, entailment of CNF queries can be computed in time polynomial in the size of the set of prime implicants plus the size of the query.

Dually one can take the conjunction of all prime implicates $C_1, \dots, C_k$ of a knowledge base KB to obtain a CNF formula $C_1 \wedge \dots \wedge C_k$ which is equivalent to KB [Reiter and deKleer, 1987]. For every CNF query Q, $KB \models Q$ iff for each nontautologous clause C' of Q there is a prime implicate C of KB such that $C \models C'$, i.e., $C \subseteq C'$. Hence, the entailment of CNF queries can be computed in time polynomial in the size of the set of prime implicates plus the size of the query.

Intuitively all prime implicates of a CNF knowledge base can be found by resolving clauses (each revolvent is an implicate) and discarding implicates which are not prime. However, this method may require too many resolution steps. Research on algorithms for computing prime implicates already started a long time ago and can be found for example in [Tison, 1967, Jackson and Pais, 1990, deKleer, 1992, Simon and delVal, 2001].

However, the number of prime implicants and prime implicates of a knowledge base with n variables were shown in [Chandra and Markowsky, 1978] to be exponential in n in the worst-case. An experimental study of the number of prime implicants and prime implicates for CNF knowledge bases of increasing size was performed in [Schrag and Crawford, 1996b]. We will discuss these results in more detail in Chapter 7 where we will use them to clarify some of our experimental results.

5.3.2 Unit-resolution-complete compilation

Since the number of prime implicates can be exponential an enhanced method was proposed in [delVal, 1994]. The method is based on the observation that entailment of a CNF query and a prime-implicates-compiled knowledge base can be done by checking whether each query clause is contained in a prime implicate. This check for containment is a form of *unit-resolution* refutation, which is defined as follows:

Definition 5.3.2.1 *Unit resolution is a form of resolution where at least one of the two clauses to be resolved is a single literal.*

Unit resolution is sound, but incomplete, i.e., not all refutations can be found by unit resolution. Negating a clause in the query yields a set of negated literals, and by unit resolution one obtains the empty clause if and only if there is a prime implicate made by a subset of the literals in the clause.

By substituting the set-containment check with unit resolution refutation, one does not need to keep all prime implicates, but only the subset of prime implicates from which each prime implicate can be derived by unit resolution. Since every unit resolution refutation needs polynomial time in the size of the initial clauses to be refuted, this method also turns a coNP-complete method into a problem solvable in polynomial time with respect to the size of the formula produced by pre-processing.

In [delVal, 1994] cases are given in which unit refutation can discard an exponential number of prime implicates. However, the method is limited to CNF knowledge bases (i.e., although any formula can be translated into an equivalent CNF formula, this may increase its size exponentially).

5.3.3 Theory prime implicates

Another method was developed by Marquis [Marquis, 1995]. He starts observing that prime implicants and prime implicates methods are based on transforming the problem $KB \models Q$ (Q being a clause), involving the entire knowledge base KB, into *local* tests involving one prime implicant/implicate at a time. He proposes to enhance such local tests with a theory, while keeping its complexity to be polynomial-time.

Definition 5.3.3.1 *A* theory prime implicate *of a knowledge base KB with respect to a theory Φ is a clause C such that $KB \cup \Phi \models C$ and for each other clause C' if $KB \cup \Phi \models C'$ and $\Phi \cup C' \models C$ then also $\Phi \cup C \models C'$.*

The theory prime implicates of a knowledge base KB with respect to a theory Φ will be denoted by TPI(KB,Φ). Note that when Φ is an empty knowledge base one obtains the definition of prime implicate, i.e., TPI(KB,$\varnothing$) = PI(KB). Hence, theory prime implicates extend prime implicates.

Observe that checking $\Phi \cup \{C\} \models C'$ is equivalent to check, for each literal $l_i \in C$, whether $\Phi \models \neg l_i \cup C'$. The key point is that *if deduction in the theory Φ can be done in* *polynomial time* entailment of CNF queries can be computed in time polynomial in the size of the set of theory prime implicates.

Marquis suggests as good candidates for Φ the set of all Horn clauses of KB, the set of all binary clauses of KB, and many others. In general, any subset of KB such that entailment is tractable can be used. However, for a knowledge base KB and two theories Φ and Φ' such that $KB \models \Phi'$ and $\Phi' \models \Phi$ holds we have that the number of clauses of TPI(KB,Φ) can never be larger than TPI(KB,Φ') (Corollary 2 in [Marquis, 1995]). Hence, to get the best theory prime implicate compilation we only have to consider the largest subsets (with respect to set inclusion) of a knowledge base KB.

Marquis gives examples in which the number of theory prime implicates is exponentially smaller than the number of del Vals' filtered prime implicates. Furthermore, in [Marquis and Sadaoui, 1996] Marquis and Sadaoui give an algorithm for computing theory prime implicates, which is based on Binary Decision Diagrams. With this algorithm, the initial knowledge base does not need to be in CNF, and the prime implicates of KB need not be generated. This reduces considerably compilation operations, as shown in some experiments.

In summary, the three classes of exact knowledge compilation methods given in the beginning of Section 5.3 are ordered according to the effectiveness of the method. Each method is at least as good as its predecessors, and for each method there exists a theory that can be compiled exponentially smaller than when using one of the earlier described knowledge compilation methods. However, according to a theorem in [Selman and Kautz, 1996] it is highly unlikely that one of the described methods (or any exact knowledge compilation method) can compile every theory into a polynomial data structure.

To overcome some of the drawbacks of exact knowledge compilation methods some of the requirements might be weakened. We will look at some of these methods in the following section.

5.4 Approximate knowledge compilation

We gave some formal definitions in Section 5.2 for knowledge compilation. However, it is not always possible to make on-line reasoning more efficient in all cases. For some important problems the requirements mentioned in Section 5.2 are unlikely to be achieved.

Two classical approaches have been developed for addressing the computational hardness of reasoning problems, which are *language restriction* and *theory approximation*. Traditionally these methods have been used for achieving tractability in reasoning problems in which there is no distinction between an on-line and off-line part. In the context of knowledge compilation, they can be described as follows:

Language restriction. The language used to represent knowledge in either the fixed or variable part may be restricted such that one can still represent interesting cases and one can compile the resulting problems.

Theory approximation. One can give up the soundness or completeness in the answer to the original reasoning problem. This means that either certain information is lost in the off-line reasoning phase, or that the theory is compiled into an equivalent theory and the soundness or incompletenes in the answer is lost in the on-line reasoning phase.

Note that in both approaches the goal is to have fast on-line reasoning.

The language restriction has not been considered often in the knowledge compilation setting. On the other hand, the theory approximation approach, which we will discuss in more detail, has had some success.

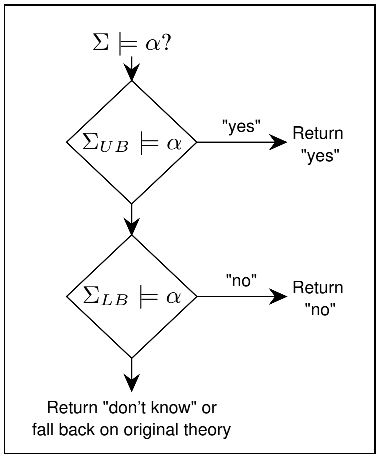

$\Sigma \models \alpha$?
$\Sigma_{UB} \models \alpha$
"yes"
Return "yes"
$\Sigma_{LB} \models \alpha$
"no"
Return "no"
Return "don't know" or fall back on original theory

Figure 5.1: Fast querying using theory approximations.

The theory approximation approach is analogous to optimization problems. In both cases, we are interested in approximate solutions that are meaningful. However, in Knowledge Representation we are not dealing with numerical problems. This means there is no obvious metric that tells us “how far we are” from the right answer to an entailment problem. The approximation of the answer to an entailment problem should therefore be grounded on a semantical basis.

The underlying idea in approximate knowledge compilation is that answers to a query can be approximated from two sides. Either with an answer that is sound but incomplete or with an answer that is complete but unsound. A sound-but-incomplete answer approximates the correct answer as a ‘yes’ answer is correct while a ‘no’ answer is in fact a ‘don’t know’. A complete-but-unsound answer approximates the correct answer as a ‘no’ answer is correct while a ‘yes’ answer is in fact a ‘don’t know’. Obviously, in both cases one wants to have an approximation that can be computed using fewer resources.

The ideas mentioned above can be formalized as follows. An approximation A of a knowledge base Σ is *sound* when for every query Q, if $A \models Q$ then $\Sigma \models Q$. In this case A is called an *upper bound* for Σ. Observe that A is an upper bound (UB) for Σ if and only if $\Sigma \models A$ holds. Dually, an approximation B of a knowledge base Σ is *complete* when for every query Q, if $B \not\models Q$ then $\Sigma \not\models Q$. In this case, B is called a *lower bound* (LB) for Σ, and $B \models \Sigma$ holds.

The approximations can be used to improve the efficiency of query answering. Suppose we have a knowledge base Σ and we want to determine if a formula α is implied by the knowledge base Σ. This can be done as depicted in Figure 5.1 where Σ_{UB} is an upper bound of Σ and Σ_{LB} is a lower bound of Σ. First, the system tries to answer the query quickly by using the approximations. If $\Sigma_{UB} \models \alpha$ then it returns ‘yes’, or if $\Sigma_{LB} \not\models \alpha$

then it returns 'no'. In case no answer is obtained, the system could simply return 'don't know', or it could decide to spend more time and use a general inference procedure to determine the answer directly from the original theory. In the latter case, the approximations could still be used to prune the search space of the inference procedure. For example, in [Kautz and Selman, 1994] queries are answered using a knowledge base Σ and also answered using the knowledge base Σ conjoined with its unit LUB. The latter is shown to speed up the query answering in their experimental setup.

5.4.1 Anytime versions of exact compilation methods

Any of the exact knowledge compilation methods discussed previously in Section 5.3 can be turned into an approximate method by stopping it before it is completed, because these methods are anytime algorithms. In fact, we can be a bit more precise about the approximations of some algorithms.

The methods based on implicates as del Val's [delVal, 1994] and Marquis' [Marquis, 1995], yield upper bounds when stopped before the entire compilation is finished. As for each implicate C by definition $\Sigma \models C$ holds, it also holds that $\Sigma \models PI_n(\Sigma)$, with $PI_n(\Sigma)$ denoting all implicates computed after n steps of one of the algorithms described in Section 5.3. Hence $PI_n(\Sigma)$ is an upperbound of Σ.

The methods computing implicants like Schrag's [Schrag and Crawford, 1996a], yield lower bounds when stopped before the entire compilation is finished. As for each implicant D of Σ it holds by definition that $D \models \Sigma$, it follows that whenever $D \not\models Q$ for each already computed implicant and for some query Q that $\Sigma \not\models Q$. Hence, the computed implicants form a lowerbound of the theory Σ.

5.4.2 Horn approximations

An original method to approximate knowledge bases was developed in [Selman and Kautz, 1991] and extended in [Selman and Kautz, 1996]. The idea is to compile a knowledge base into a formula which belongs to a syntactic class which guarantees polynomial-time inference.

In the method developed in [Selman and Kautz, 1991] a knowledge base is approximated by a Horn formula. The basic idea is to bound a set of models of the original theory from below (i.e., complete) and from above (i.e., sound) which is formalized in the following definition.

Definition 5.4.2.1 *Let Σ be a set of clauses. The set Σ_{LB} and Σ_{UB} of Horn clauses are respectively a Horn lower bound and a Horn upper bound of Σ if and only if*

$$\mathcal{M}(\Sigma_{LB}) \subseteq \mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma_{UB})$$

or, equivalently,

$$\Sigma_{LB} \models \Sigma \models \Sigma_{UB}.$$

Instead of using any pair of bounds to characterize the original theory, we would like to use the best possible bounds. This leads to a *greatest* Horn lower bound and a *least* Horn upper bound.

Definition 5.4.2.2 *Let Σ be a set of clauses. The set Σ_{GLB} of Horn clauses is a greatest Horn lower bound of Σ if and only if $\mathcal{M}(\Sigma_{GLB}) \subseteq \mathcal{M}(\Sigma)$ and there is no set Σ' of Horn clauses such that $\mathcal{M}(\Sigma_{GLB}) \subset \mathcal{M}(\Sigma') \subseteq \mathcal{M}(\Sigma)$.*

Definition 5.4.2.3 *Let Σ be a set of clauses. The set Σ_{LUB} of Horn clauses is a least Horn upper bound of Σ if and only if $\mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma_{LUB})$ and there is no set Σ' of Horn clauses such that $\mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma') \subset \mathcal{M}(\Sigma_{LUB})$.*

Each theory has a unique LUB (up to logical equivalence), but can have many different GLBs.

As shown in Figure 5.1, inference can be approximated by using the Horn GLBs and Horn LUBs. In this way, inference could be unsound or incomplete, but it is always possible to spend more time and use a general inference procedure on the original theory.

Similar to Horn bounds other bounds can also be used. In [Selman and Kautz, 1996] the GLB and LUB are computed, which only contain unit clauses (i.e., substitute in the definitions above unit clause for Horn clause). Their experimental results show that such a restricted language for the bounds can already lead to a substantial savings in computation time.

5.5 Conclusions

We have given an overview of knowledge compilation, which can be used to address the computational complexity of logical inference. The central idea behind knowledge compilation is the observation that many problems can conceptually be split into two parts: a fixed part and a variable part. Several knowledge compilation methods have been described. These methods were divided into exact knowledge compilation methods and approximate knowledge compilation methods.

In the next chapter we will look at planning problems, which belong to the problems that can be divided into a fixed part and a variable part. Thereafter, in Section 7, we will give an experimental analysis of some of the knowledge compilation methods described in this chapter on various planning problems.

Chapter 6

Planning

6.1 Introduction

Artificial Intelligence Planning is concerned with choosing a sequence of actions that will achieve certain goals. The basic description of a planning algorithm has remained constant for nearly three decades: Given the input, which consists of

1. a description of the *initial state* of the world,
2. a description of the *goals* to be achieved, and
3. a description of the possible actions that can be performed (i.e., *domain theory*),

compute a sequence of actions which, when executed in any world that satisfies the initial state, will achieve the goals.

The formulation of a planning problem as given above is quite abstract as it really specifies a *class* of planning problems parameterized by the languages used to represent the world, goal, and actions. In this dissertation, we will restrict ourselves to one of the earlier representational methods for planning domains that is known as STRIPS [Fikes *et al.*, 1971].

In the following section we describe the STRIPS representation in more detail. Thereafter we will describe in Section 6.3 how planning can be tranlated to a satisfiability problem. This will allow us to apply the techniques from Chapter 5 to study the application of knowledge compilation methods to planning problems in Chapter 7. However, there are various ways to encode a planning problem as a satisfiability problem. The rest of this chapter (Section 6.3.1) gives a framework for the various encodings that can be used to encode a planning problem as a satisfiability problem. Furthermore, some optimizations are given in Section 6.3.2 for the various encodings in the framework.

6.2 The STRIPS representation

The STRIPS representation describes the initial state of the world with a complete set of ground literals. The goals are defined as a propositional conjunction and restricted to goals of attainment. Furthermore, all world states satisfying the goal formula are considered equally good. The actions in the domain theory are described with a conjunctive precondition and conjunctive effect that define a transition function from worlds to worlds. The action can be executed in any world w satisfying the precondition formula. The result of executing an action in world w is described by taking w's state description and adding each literal from the action's effect conjunction in turn, eliminating contradictory literals along the way.

6.3 Planning as satisfiability

There has been a widespread belief in the AI community that planning problems could only be solved by special purpose planning algorithms. However, improvements in the last decade on planning as satisfiability have cast this belief in doubt [Kautz and Selman, 1992, Kautz and Selman, 1996].

The architecture of a typical SAT-based planning system is shown in Figure 6.1. The *encoder* takes a planning problem as input, which consists of the initial state, goal and domain theory written in some formal language, and generates a propositional theory in Conjunctive Normal Form based on a guessed length for the plan, which is usually one. (Note that the length of the plan is necessary to determine the alphabet for the encoding.) Furthermore, a symbol table is generated, which records the correspondence between propositional variables and the planning instance. Because the theory is encoded in such a way that every satisfying assignment corresponds to a successful plan it can be given to a *SAT solver*. If the SAT solver finds that the propositional theory is unsatisfiable, the compiler is called to generate a new encoding based on a plan length, which is larger than previous attempts (usually the length of the plan is incremented by one). Finally, if the SAT solver finds a model of the propositional theory this is given to a *Decoder*. The decoder uses the symbol table to translate the model into a plan that can be used by a human user.

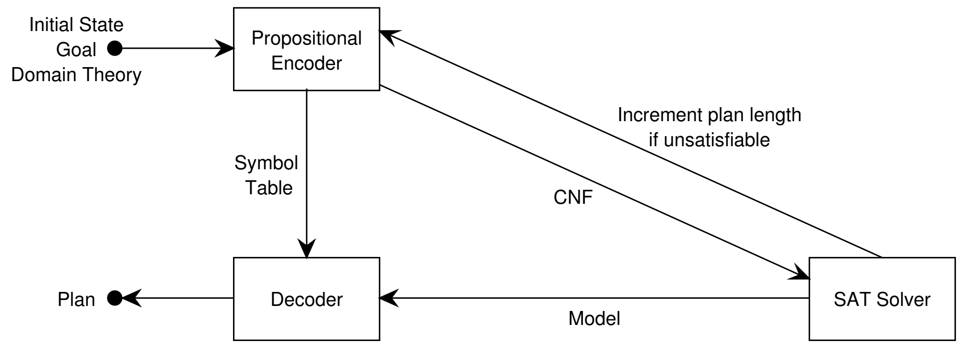


Figure 6.1: Architecture of a typical SAT-based planning system.

6.3.1 The space of encodings

It is well known that the representation of a problem can have a major impact on the efficiency of the algorithms that try to solve that particular problem. Since a systematic SAT solver may in the worst case take time exponential in the size of the input theory, one wants to make the theory created by the encoder as small as possible. For logical theories, the measure of the size of the theory is complicated because it can be measured in several ways (e.g., number of variables, number of clauses, total number of literals summed over all clauses). The decrease in one parameter usually increases another. To support the analysis of various encodings [Ernst *et al.*, 1997] developed a parameterized space with two dimensions:

1. *Action representation*: This choice specifies the correspondence between propositional variables and ground (fully-instantiated) plan actions. Options are a regular, simply split, overloaded split, or bitwise action representation. These choices represent different points in the tradeoff between the number of variables and the number of clauses in the formula.
2. *Frame axioms*: The choice of classical or explanatory frame axioms varies the way that stationary fluents are constrained.[i]

Each of the encodings uses a standard fluent model in which time takes nonnegative integer values. State-fluents occur at even-numbered times and actions at odd times. All of the encodings use the following set of *universal axioms*:

INIT: The initial state is completely specified at time zero, including all properties presumed by the closed world assumption.

GOAL: In order to test for a plan of length n, all desired goal properties are asserted to be true at time $2n$.

A⇒P,E: Actions imply their preconditions and effects. For each odd time t between 1 and $2n-1$ and for each consistent ground action, an axiom asserts that execution of the action at time t implies that its effects hold at $t+1$ and its preconditions hold at $t-1$.

6.3.1.1 Dimension: action representation

For the action representation one has to choose whether the names of ground action instances should be represented in regular, simply split, overloaded split, or the bitwise format. The choice is irrelevant for purely propositional planning problems, but can be crucial for parameterized action schemata.

As systematic solvers may need exponential time in the number of variables, one wants to reduce this number. In the *regular* representation a logical variable represents a ground action. Different ground actions are represented by different logical variables. This leads

[i]A fluent is an instantiated predicate, e.g., `at(pkg,loc1,time1)` in the Logistics domain.

to an encoding containing $n|Ops||Dom|^{A_o}$ variables (The symbols are defined in Figure 6.2). The other options for the action representation are methods developed to reduce the number of variables in an encoding.

With *simple action splitting*, which was developed in [Kautz and Selman, 1996], each n-ary action fluent is represented by n logical variables in the encoding. For example, `Move(A,B,C,t)` is replaced by the conjunction of `MoveArg1(A,t)`, `MoveArg2(B,t)`, `MoveArg3(C,t)`. (Note that each conjunct is in fact represented by a propositional variable, not a functional term. The notation used here, is to emphasize the combinatorics and clearly represent the effect of various encodings on the CNF size.)

In simple splitting propositional variables are only shared between instances of the same operator. Another method, called *overloaded splitting*, also allows the sharing of propositional variables between instances of different operators. More precisely, all operators share the same split fluents. For example, overloaded splitting replaces `Move(A,B,C,t)` by the conjunction of `Act(Move,t), Arg1(A,t), Arg2(B,t), Arg3(C,t)`, while a different action `Paint(A,Red,t)` is replaced with `Act(Paint,t), Arg1(A,t), Arg2(Red,t)`.

In the *bitwise* encoding a number of bit symbols are added to the theory and all ground action instances are numbered. The number encoded by the bit symbols determines the ground action, which executes at each odd time step. For example, if there were four ground actions then (¬ `bit1(t)` ∧¬ `bit2(t)`) would replace the first action, (¬ `bit1(t)` ∧ `bit2(t)`) would replace the second, and so forth.

The size of the number of variables for each action representation is summarized in Figure 6.2. Note that they are in decreasing order. The effect of the various encodings on the number of variables has been investigated in [Ernst *et al.*, 1997]. The results of [Ernst *et al.*, 1997] are prelimanary results, but indicate that the regular and simply split representations are good choices. Although, the regular and simply split representations contain more variables than the overloaded and bitwise representations (Figure 6.2), this no longer holds after simplifying the various representations. The bitwise and overloaded representations result in extremely complex encodings that resist simplification. For example, in [vanGelder and Tsuji, 1996] a linear-time procedure is described for simplifying the representations and they show that bitwise representations, which initially contained the smallest number of propositional variables, afterwards contained the highest number of propositional variables.

6.3.1.2 Dimension: frame axioms

The second major choice that has to be made is about the axioms that deal with the frame problem. The frame axioms constrain unaffected fluents when an action occurs. There are two alternatives: classical or explanatory frame axioms.

Classical frame axioms [McCarthy and Hayes, 1969] state which fluents are left unchanged by a given action. For example, in the Blocksworld planning problem moving a block A from B to C leaves intact whether block D is clear. With classical frame axioms this would be represented as

$$Clear(D, t-1) \wedge Move(A, B, C, t) \Rightarrow Clear(D, t+1).$$

The universal axioms together with the classical frame axioms almost produces a valid encoding of the planning problem. The encoding is not valid as it does not enforce some action to occur at each odd time t. If no action occurs at time t, the axioms of the encoding can infer nothing about the truth-value of fluents at time $t+1$, which can therefore take on arbitrary values. The encoding can be made valid by adding AT-LEAST-ONE axioms for each time step.

AT-LEAST-ONE: A disjunction of every possibly, fully-instantiated action ensures that some action occurs at each odd time step. (A maintenance action is inserted as a pre-processing step.)

The resulting plan consists of a totally ordered sequence of actions. If more than one action occurs at one time step either one can be selected to form a valid plan. This follows from the $A \Rightarrow P, E$ and classical frame axioms, because any two actions occurring at time t lead to an identical world-state at time $t+1$.

Explanatory frame axioms [Haas, 1987] enumerate the set of actions that could have occurred in order to account for a state change. For example in the Blocksworld planning problem, if the status of a block D changes from clear to not clear this can be represented by an explanatory frame axiom as follows:

$$Clear(D, t-1) \wedge \neg Clear(D, t+1) \Rightarrow (Move(A, B, D, t) \vee$$
$$Move(A, C, D, t) \vee \ldots \vee Move(C, Table, D, t)).$$

The precondition states that the status of D is changed from clear to not clear at time t. The postcondition enumerates all actions that could have caused this change, which are all possible movements of blocks (different from D) on top of D. Adding the explanatory frame axioms to the universal axioms leads to a reasonable encoding (i.e., no other axioms are necessary) [Ernst *et al.*, 1997].

Since explanatory frame axioms do not explicitly force the fluents not affected by an executing action to remain unchanged they bring an important benefit: *parallelism*. To

Action Representation	Nr. of Variables
regular	$n\|Ops\|\|Dom\|^{A_o}$
simple splitting	$n\|Ops\|\|Dom\|A_o$
overloaded splitting	$n(\|Ops\| + \|Dom\|A_o)$
bitwise	$\lceil log_2(\|Ops\|\|Dom\|^{A_o})\rceil$

Figure 6.2: The number of variables for each action representation. $|Ops|$ is the number of operators; $|Dom|$ is the number of constants in the domain; n is the number of odd time steps in the plan; A_o is the maximal arity of operators.

guarantee a linear plan or a plan that can easily be transformed into a linear plan one can add EXCLUSION axioms.

EXCLUSION: Transforming a plan into a linear plan is guaranteed by restricting which actions may occur simultaneously.

Two kinds of exclusion axioms enforce different constraints in the resulting plan.

1. *Complete* exclusion: For each odd time step, and for all distinct, fully-instantiated action pairs α, β, add clauses of the form $\neg\alpha_t \vee \neg\beta_t$. In this way a totally ordered plan is guaranteed as complete exclusion ensures that only one action can occur at each time step.
2. *Conflict* exclusion: For each odd time step, and for all distinct, fully-instantiated, conflicting action pairs α, β, add clauses of the form $\neg\alpha_t \vee \neg\beta_t$. Two actions conflict if one's precondition is inconsistent with the other's effect. Conflict exclusion results in plans whose actions form a partial order. Any total order consistent with the partial order is a valid plan.

The two kinds of exclusion axioms cannot be used with every action representation. In simple splitting there is no unique variable for each fully-instantiated action.

The bitwise representation requires no action EXCLUSION axioms. At any time step, only one fully-instantiated action's index can be represented by the bit symbols, so a total ordering is guaranteed.

6.3.2 Optimizations

The automatic encodings from the parameterized space discussed above can be improved in several ways. The methods that will be discussed here are compile-time optimizations like factoring, and the addition of domain specific information.

6.3.2.1 Factoring

Some choices that lead to a small number of variables (i.e., splitting strategies and bitwise) tend to explode the number of clauses or size of each clause. For example, this can happen when using the AT-LEAST-ONE axioms, which is a disjunction of all fully-instantiated actions. Substituting a conjunction of split or bitwise variables for each regular action literal produces a DNF formula that blows up exponentially when converted to CNF.

Factoring can reduce both the number of clauses and their sizes for simple and overloaded splitting. The idea is to use only a subset of the full conjunction for an action whenever possible. Such a partially-instantiated action represents the set of all fully-instantiated actions consistent with it. The bitwise action representation is unsuitable for factoring because partial conjunctions of bit variables are not useful.

6.3.2.2 Domain specific information

The addition of domain specific information is another way to optimize the encoding generated by the compiler. Typically this knowledge is impossible to express in terms of STRIPS actions but can easily be given in general logical axioms. Some of the domain specific knowledge could be given by a user, while other information could be induced automatically by processing the action schemata and initial state specifications.

Domain axioms may be classified in terms of the logical relationship between the knowledge encoded and the original problem statement [Kautz and Selman, 1998]:

1. Action *conflicts* and *derived effects* are entailed solely by the preconditions and effects of the domain's action schemata.
2. Heuristics, which are entailed by the operator and initial state axioms together, include *state invariants*. For example, in the Logistics domain moving a vehicle from one location to another does not entail that a truck is always at a single location. However, if in the initial state every truck is at exactly one location, the operator will propagate that invariant to all future states.
3. *Optimality* heuristics restrict plans by disallowing unnecessary subplans. For example, in the Logistics domain one could add the rule "do not return a package to a location from which it has been removed".
4. *Simplifying* assumptions are not logically entailed by the domain theory and therefore introduce incompleteness into the encoding. However, in many cases assumptions can be made without transforming a solvable instance into an inconsistent one. For example, in the Logistics domain one could add "once a truck is loaded, it should immediately move".

[Ernst *et al.*, 1997] shows that the addition of domain specific knowledge of the types listed above increases the clause-size of the resulting CNF formulae, but decreases the number of variables (after simplification) to as much as 15% which speeded solver time significantly.

6.4 Conclusion

We introduced planning and the STRIPS language for its representation. The rest of this chapter dealt with a framework for the various encodings that can be used to represent planning as a satisfiability problem. The framework is a parameterized space with two dimensions: the action representation and the frame axioms. The choice for the action representation determines the number of variables needed in the encoding. Furtermore, some optimizations were given to shrink the size of the encoding even further, i.e., factorization and additional domain specific knowledge.

Preliminary results suggest that for the action representation the regular and simply split representations are good choices [Ernst *et al.*, 1997]. In contrast, the bitwise and overloaded representations result in complex encodings that resist simplification and type analysis. For the frame axioms, results by [Kautz and Selman, 1996, Ernst *et al.*, 1997] show that explanatory frame axioms are clearly superior to classical frames in almost every case. Another benefit is the parallelism permitted when used in combination with conflict exclusion.

In the next chapter we will perform an empirical analsysis of knowledge compilation techniques (discussed in Chapter 5) on planning problems. For our analysis we will use the framework discussed in this chapter and analyse the properties of the various encodings it contains.

Chapter 7

Applying knowledge compilation

7.1 Introduction

The computational intractability of propositional reasoning is a fundamental issue in Knowledge Representation. In the last decade knowledge compilation has emerged as a research direction for dealing with this problem [Selman and Kautz, 1991, Kautz and Selman, 1991]. The goal of knowledge compilation is to reduce the complexity of the on-line computation by pre-processing a part of the problem in an off-line phase. Algorithms for knowledge compilation have improved considerably since the first proposals based on implicates. For example, Marquis has developed a method based on prime implicates [Marquis and Sadaoui, 1996] and [Simon and delVal, 2001] has developed a method based on kernel resolution. Also guidelines are being developed for using knowledge compilation techniques. In [Darwiche and Marquis, 2001] guidelines are given for choosing the target language. It is believed that knowledge compilation is mature for applications [Cadoli and Donini, 1997]. This belief is supported by empirical analysis reporting positive results [Kautz and Selman, 1994, Schrag and Crawford, 1996a, Simon and delVal, 2001]. On the other hand, theoretical results indicate that knowledge compilation has some serious limitations. It is well known that it is highly unlikely that there exists an exact knowledge compilation approach that can compile any given knowledge base into an appropriate polynomial data structure [Selman and Kautz, 1996]. Also many interesting problems (e.g., diagnosis, planning) have been proven to be incompilable [Liberatore, 1998].

Although knowledge compilation techniques have been improving it is still not clear what can and what cannot be achieved by these methods. On the one hand positive results have been obtained, but it is not clear if these can be extended to other problem domains. On the other hand negative results have been obtained, but some of these results are based on a worst-case complexity analysis and therefore provide little information for typical problems found in practice. (To support this, take for example the regular action representation for planning problems discussed in Section 6.3.1.1. Contrary to worst-case complexity results [Ernst *et al.*, 1997] showed that the regular action representation was surprisingly effective in practice.) To make the boundaries of knowledge compilation more clear, empirical analyses of current knowledge compilation techniques on various problem domains should be performed. We believe that this area of research is still limited.

To investigate the practical value of knowledge compilation some knowledge compilation methods are analysed by applying them to planning problems. Planning is a notoriously hard problem [Erol *et al.*, 1995, Cadoli, 1993] which can be divided into a fixed part (domain and operators) and a variable part (initial state and goals) and is therefore a prime candidate for knowledge compilation. Furthermore, techniques exist to translate planning problems from some planning description language into a propositional theory, for which compilation techniques are available.

The rest of this chapter is divided as follows. Section 7.2 discusses how to adjust the planning as satisfiability approach such that knowledge compilation techniques can be applied. This leads to a process containing several components. Section 7.3 describes in more detail which choices need to be made for the various components. Section 7.4 gives the motivation for the choices made in the empirical analysis reported in this chapter. After that related work from the literature is described in Section 7.5 to give the context of our work. Section 7.6 gives the results of the empirical analysis of knowledge compilation on planning problems. This section is the largest of this chapter and divides into three subsections. Each subsection deals with a specific choice that can have an influence on the performance of knowledge compilation. First, the influence different encodings can have on knowledge compilation is analysed. Second, the influence different domains can have, and finally the influence of different knowledge compilation methods. The chapter ends with conclusions (Section 7.7), and gives ideas for future work (Section 7.8).

7.2 The knowledge compilation approach to planning

In Section 6.3 planning as satisfiability is discussed and a typical architecture for a SAT-based planning system (Figure 6.1) is given. When knowledge compilation is applied to planning, an additional step needs to be added to the architecture, which is shown in Figure 7.1.

The encoding from the propositional encoder is split into two parts, *the fixed part*, which contains the domain and operator descriptions, and *the variable part*, which contains the initial state and goals. The fixed part is given to a knowledge compilation method, which compiles it into another language. The compiled fixed part together with the variable part is then given to a SAT solver. If the SAT solver finds that the propositional theory is unsatisfiable, the variable part is adjusted by setting the goals at a larger point in time. (Note that in Figure 6.1 the Propositional Encoder is called again when the SAT encoding is found to be unsatisfiable, whereas in Figure 7.1 the SAT solver is called again. This difference will be discussed in more detail in Section 7.6.1.2.) The SAT solver is called again on the adjusted variable part and the compiled fixed part until a model of the propositional theory is found. The model is then given to a decoder, which translates the model into a plan.

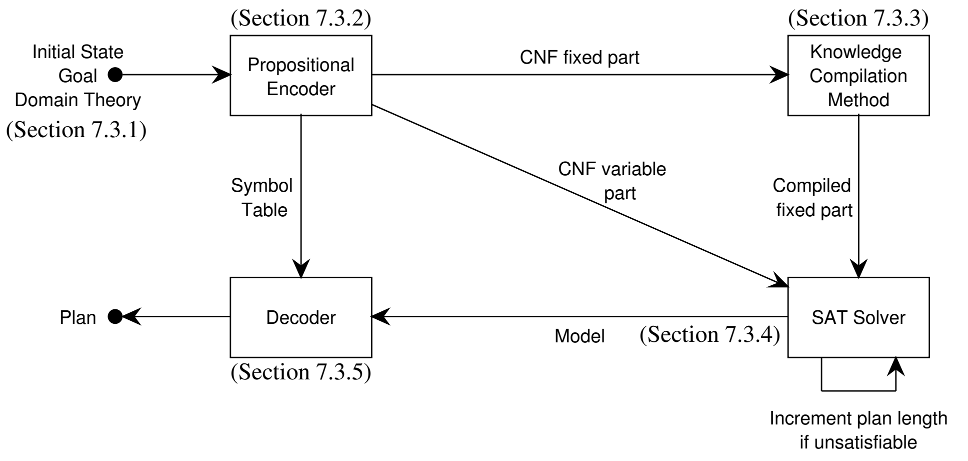


Figure 7.1: The process of performing knowledge compilation in the planning as satisfiability framework.

7.3 Choices and motivation

The architecture in Figure 7.1 describing knowledge compilation in the planning as satisfiability framework contains several components. This section describes the choices that need to be made for these components.

7.3.1 Planning problems

The planning problems used in the empirical analysis come from a set of benchmark problems used in the Artificial Intelligence Planning Systems (AIPS) competition.[i] The problems are written in the Planning Domain Definition Language (PDDL, [Ghallab *et al.*, 1998]), which is the standard language for encoding planning domains. However, any formal language can be used as long as there is a means to translate the planning problems into propositional logic. The benchmark problems used in the empirical analysis are the Logistics planning problem, the Tower of Hanoi problem, and the Monkey problem.

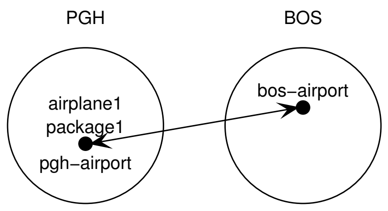
7.3.1.1 The Logistic problem

The basic elements of the Logistic problem are the *infrastructure*, the *transportation units*, and the *packets*. The infrastructure consists of a set of cities, and each city consists of a set of locations. The fact that a location `loc` is situated in a city `cit` is denoted by

[i]ftp://ftp.cs.washington.edu/pub/ai/domains-pddl.tgz

the relation `in-city(loc,cit)`. In each city there is at least one location `loc` that is also an airport, denoted by `airport(loc)`. Transportation units consist of trucks and airplanes. Trucks may move freely within a city, but airplanes between airports can only do intercity transport. The objective of the planning problem is to transport all packets from locations specified in the initial state to locations specified in the goal. A packet `pkg` can be at a location `loc`, denoted by `at(pkg,loc)`, or in a transportation unit `unit`, denoted by `in(pkg,unit)`. Each transportation unit has a set of *actions* associated to it, which can be one of the following types: (1) a *move* action to represent driving or flying from one location to another, (2) a *load* action to load a packet in a transportation unit, or (3) an *unload* action. Each transportation unit is assumed to have unlimited capacity.

One of the benchmark problems that used in the empirical analysis is shown in Figure 7.2. This benchmark problem can be solved in 3 steps.



Domain Information
```
city(PGH); city(BOS);
location(pgh-airport);
location(bos-airport);
in-city(pgh-airport,PGH);
in-city(bos-airport,BOS).
```

Initial State
at(package$_1$,pgh-airport);
at(airplane$_1$,pgh-airport).

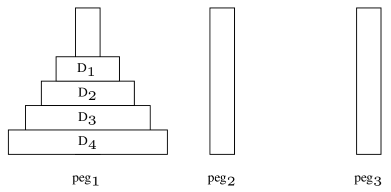
Goal Formula
at(package$_1$,bos-airport).

Figure 7.2: The att-log0a benchmark problem.

7.3.1.2 The Tower of Hanoi problem

The French mathematician Edouard Lucas invented the Tower of Hanoi puzzle in 1883. The puzzle consists of a number of disks and three pegs. Each disk has a hole in the middle, which allows it to be placed on one of the pegs. Furthermore, each disk has a different size and can only be placed on a larger disk or an empty peg. The relative size of two disks is denoted by `smaller(dsk1,dsk2)`. The objective of the puzzle is to transfer the disks from some initial position to a position specified in the goal. The standard instance of this problem is to move a tower of disks from one peg to another. The only action associated to each disk is a *move* action, which moves one disk at a time from one peg to another. The operator is only allowed when the disk to be moved has no other disks on top, denoted by `clear(dsk)`, and when the disk is moved onto a larger disk or an empty peg.

One of the benchmark problems that used in the empirical analysis is shown in Figure 7.3. This benchmark problem can be solved in 15 steps.



Domain Information
`smaller`(D_i,D_j) $i, j \in \{1,2,3,4\}, i < j$;
`smaller`(peg_i,D_j) $i \in \{1,2,3\}$,
$j \in \{1,2,3,4\}$.

Initial State
`on`(D_i,D_{i+1}) $i \in \{1,2,3\}$;
`on`(D_4,peg_1);
`clear`(X) $X \in \{d_1, peg_2, peg_3\}$;
`not clear`(D_i) $i \in \{1,2,3\}$.

Goal Formula
`on`(D_i,D_{i+1}) $i \in \{1,2,3\}$;
`on`(D_4,peg_2).

Figure 7.3: The Hanoi-strips4 benchmark problem.

7.3.1.3 The monkey problem

In the monkey problem there is a room with a monkey who wants to have some bananas, which are hanging from the ceiling. However, the bananas are out of reach for the monkey. In the room a box is available, which enables the monkey to reach the bananas if he climbs on it. Before the monkey can actually get the bananas he will have to cut them with a knife which is lying somewhere on the floor. Furthermore, there is a waterfountain and a glass. If the monkey wants to have some water, he will have to get the glass and climb the box in front of the fountain. The objective of the problem is for the monkey to get the bananas and/or the water.

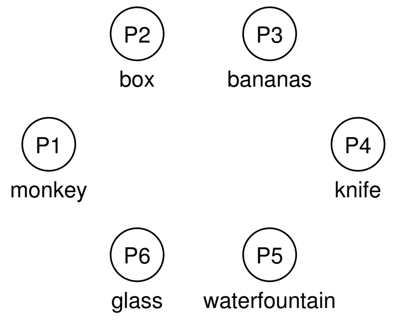
In the room several locations are identified, denoted by `location(loc)`, and all objects are assigned a location (e.g., `at(box,loc1)`). The monkey can perform the following actions: (1) a *move* action to get to another location, (2) a *push* action, to push the box to another location, (3) a *climb* action to stand on the box, and (4) a *grasp* action to get an object.

One of the benchmark problems that used in the empirical analysis is shown in Figure 7.4. This benchmark problem can be solved in 10 steps.

7.3.2 Translating planning problems

The Medic Planner [Ernst *et al.*, 1997] was used to translate the planning problems into propositional logic. The Medic planner is an automatic compiler, which accepts traditional planning inputs (initial state, goal formula, and STRIPS action schemata) and generates an encoding dependent on certain switch settings. These switch settings, which are described in Section 6.3.1, include the action representation, frame axioms, exclusion and type optimizations. By choosing different switch settings, Medic can generate up to twelve different encodings.

Each encoding generated by Medic can be characterized by its switch settings, which can be represented by a four-letter word. The four letters correspond to the following



Domain Information
`location(P`$_i$`)` $i \in \{1, \ldots, 6\}$.

Initial State
`at(monkey,P`$_1$`); onfloor;`
`at(box,P`$_2$`); at(bananas,P`$_3$`);`
`at(knife,P`$_4$`); at(glass,P`$_6$`);`
`at(waterfountain,P`$_5$`).`

Goal Formula
`has bananas;`
`has water.`

Figure 7.4: The monkey-test2 benchmark problem.

abbreviations for the flag settings:

Frames: c=classical, e=explanatory.

Action representation: r=regular, s=simple split, c=factored simple split, o=overloaded split, f=factored overloaded split, b=bitwise.

Exclusion: s=sequential pairwise, p=parallel.

Types: n=no special treatment, t=don't replicate per step, e=eliminate.

In Section 7.6, which presents the empirical analysis of knowledge compilation, this shorthand notation is used for the various encodings.

7.3.3 Knowledge compilation methods

Three knowledge compilation methods were used for the knowledge compilation proces. This section describes these methods in some detail.

7.3.3.1 Kautz and Selmans LUB approximation

The first method used is the LUB approximation algorithm of Kautz and Selman [Selman and Kautz, 1991], which is one of the earliest attempts in making an algorithm for knowledge compilation. Originally, the algorithm was used for approximating a logical theory by a Horn upper bound, but the LUB algorithm has also been proven to be correct for any target language closed under subsumption in [delVal, 1995].

The procedure to compute the $\mathcal{L}_T$-LUB for any target language $\mathcal{L}_T$ that is closed under subsumption is shown in Figure 7.5. The algorithm is a brute force resolution algorithm modified to avoid resolving pairs of clauses both of which belong to the target language $\mathcal{L}_T$.

```
Procedure Generate-$\mathcal{L}_T$-LUB($\Sigma$)

begin
$\Sigma_T := \{C \in \Sigma \mid C \in \mathcal{L}_T$ and $C$ is not tautologous$\}$
$\Sigma_N := \{C \in \Sigma \mid C \notin \mathcal{L}_T$ and $C$ is not tautologous$\}$
loop
  choose clauses $C_1 \in \Sigma_T \cup \Sigma_N$, $C_2 \in \Sigma_N$
    with a non-tautologous resolvent $C$ which
    is not subsumed by any clause in $\Sigma_T \cup \Sigma_N$.
  if no such choice is possible then exit loop endif
  if $C \in \mathcal{L}_T$
  then delete from $\Sigma_T$ and $\Sigma_N$ any clause subsumed
    by $C$; $\Sigma_T := \Sigma_T \cup \{C\}$
  else delete from $\Sigma_N$ any clause subsumed by $C$
    $\Sigma_N := \Sigma_N \cup \{C\}$
  endif
endloop
return $\Sigma_T$
end
```

Figure 7.5: Pseudocode for Kautz and Selmans LUB approximation algorithm.

7.3.3.2 Zres

The second method used was reported in [Simon and delVal, 2001]. This method is based on kernel resolution [delVal, 1999] using Zero-Suppressed Binary Decision Diagrams (ZBDDs) for its implementation. The method can be used to compute all prime implicates of a given theory and has been shown to be effective even on theories with very large numbers of prime implicates.

Kernel resolution is a consequence-finding generalization of ordered resolution. A total order is assumed of the propositional variables $x_1, \ldots, x_n$. A *kernel clause* C is a clause partitioned into two parts, the skip $s(C)$, and the kernel $k(C)$. Given any target language $\mathcal{L}_T$ closed under subsumption, a $\mathcal{L}_T$*-kernel resolution deduction* is any resolution deduction constructed as follows: (1) for any input clause C, set $k(C) = C$ and $s(C) = \varnothing$; (2) resolutions are only permitted upon kernel literals; (3) the literal l resolved upon partitions the literals of the resolvent into those smaller (the skip), and those larger (the kernel) than l, according to the given ordering; and (4) to achieve focussing, any resolvent R is required to be $\mathcal{L}_T$-acceptable, which means that $s(R) \in \mathcal{L}_T$.

In order to search the space of kernel resolution proofs, to each variable x_i a bucket $b[x_i]$ is associated of clauses containing x_i. The clauses in each bucket are determined by an indexing function $I_{\mathcal{L}_T}$, so that $C \in b[x_i]$ iff $x_i \in I_{\mathcal{L}_T}(C)$.

Bucket elimination (BE), is an exhaustive search strategy for kernel resolution. BE processes buckets $b[x_1], \ldots, b[x_n]$ in order, computing in step i all resolvents that can be obtained by resolving clauses of $b[x_i]$ upon x_i, and adding them to their corresponding buckets, using $I_{\mathcal{L}_T}$. The algorithm, which uses standard subsumption policies, is complete for finding consequences of the input theory which belongs to the target language $\mathcal{L}_T$, that is, BE$(\Sigma) \cap \mathcal{L}_T = PI_{\mathcal{L}_T}(\Sigma)$.

All prime implicates can be computed by setting the target language $\mathcal{L}_T$ mentioned above to be full propositional logic. As is shown in [delVal, 1999], BE is in this case identical to Tison's prime implicate algorithm [Tison, 1967].

A **ZBDD** is an extension of a binary decision diagram (BDD), which is a directed acyclic graph with a unique source node, only two sink nodes (1 and 0, interpreted as true and false respectively) and with labeled nodes $\Delta(x, n_1, n_2)$. Such a node x has only two children (n_1 and n_2, connected respectively to its 1-arc and 0-arc) and is classically interpreted as the function $f = if\ x\ then\ f_1\ else\ f_2$ with f_1 and f_2 the functions that interpret the BDDs n_1 and n_2 respectively.

The power of a BDD comes from the reduction rules that can be applied on it. A reduced ordered BDD (ROBDD) is a BDD in which the variables are sorted according to a given order and does not contain any isomorphic subgraph (*node-sharing* rule). In addition, the *node-elimination* rule deletes all nodes $\Delta(x, n, n)$ that do not care about their values. With the classical semantics, each node is labeled by a variable and each path from the source node to the 1-sink represents a model of the encoded formula. Hence, the ROBDD can be viewed as an efficient representation of all models of a formula.

In order to use the compression power of BDDs for encoding sparse sets instead of just Boolean functions, [Minato, 1993] introduced ZBDDs. The principle is to encode the Boolean characteristic function of a set. For this purpose, Minato changed the *node-elimination* rule into the Z*-elimination* rule for which useless nodes are those of the form $\Delta(x, 0, n)$. So, if a variable does not appear on a path, then its default interpretation is now *false*. If one wants to encode only sets of clauses, each ZBDD variable needs to be labeled by a literal of the initial formula, and each path to the 1-sink now represents the clause which contains only the literals labeling the parents of all 1-arcs of this path.

The usefulness of ZBDDs is illustrated by the fact that there exist theories with an exponential number of clauses that can be captured by ZBDDs of polynomial size. Therefore, operators can be designed which can handle sets of clauses efficiently, which depends only on the size of the ZBDD and not on the size of the encoded set.

7.3.3.3 Directional resolution

The third method used is Directional Resolution [Dechter and Rish, 1994], which is a variation of the original Davis and Putnam algorithm [Davis and Putnam, 1960]. In addition to determining the satisfiability of a theory, the algorithm generates an equivalent theory that facilitates model generation and processing of queries. Directional Resolution can therefore be seen as a knowledge compilation algorithm.

The algorithm can be described as follows. Given an arbitrary ordering of the propositional variables, each clause is assigned the index of the highest ordered literal in that clause. Then the clauses are resolved having the same index and only on their highest literal.

The algorithm can also include additional steps like forcing unit resolution, preferring resolution over literals that appear only positively or only negatively, or subsumption elimination. However, the elimination of highest indexed variables is the core of the algorithm.

This core can also be seen as a bucket elimination algorithm. Given a variable ordering

$Q_1, \ldots, Q_n$, the bucket for Q_i $bucket_i$ contains all clauses Q_i that do not contain any symbol higher in the ordering. Directional Resolution then processes the buckets in reverse order. When processing $bucket_i$, it resolves over Q_i all possible pairs of clauses in the bucket and inserts the resolvents into the appropriate lower buckets.

7.3.4 Constructing a model

Planning can be solved as a satisfiability problem because a planning problem can be encoded in such a way that every model of the propositional encoding corresponds to a successful plan. This also holds for the planning problems generated by the Medic planner. A plan can be found by first finding a model using some SAT solver and translate the model afterwards into a plan. The SAT solver is used on the compiled fixed part of the planning problem together with the variable part. This variable part is iteratively adjusted by setting the goals at larger points in time. In other words, the SAT planner is asked to answer the question "is there a plan of length n?" where n is increased by one after a negative answer. In this way, the shortest solution will always be found.

It should be clear that any SAT solver could be used for this component in the architecture. For the empirical analysis zChaff was used, as this was one of the fastest SAT solvers at the time of writing.

7.3.5 Translating the model into a plan

The last step in the process is a translation of the model found by the SAT solver into a form, which is readable for a human user. This can easily be accomplished by writing a small program in some programming language.

7.4 Motivations for our experimental setup

An empirical analysis is only useful if the experimental setup is well chosen and the motivation behind certain choices are made clear. This section gives the motivations for choosing the prime implicates as target compilation language.

In [Darwiche and Marquis, 2001] several dimensions are given which can be used to choose between different target languages for knowledge compilation. These dimensions include the succinctness of the target language, the type of queries one wants to solve with the compiled theory, and the transformations you want to perform on the compiled theory.

In the planning as satisfiability framework, the final plan is constructed from a model given by some SAT solver. Hence, the target language should be able to generate a model in polynomial time.

Furthermore, after the compilation a variable part is added before it is processed by some SAT solver. This adding of the variable part is a transformation of the compilation and should be possible in the chosen target language.

Finally, from the languages that satisfies the above two criteria a language should be chosen that is the most succinct.

Although there are more target languages possible than the fifteen sublanguages discussed in [Darwiche and Marquis, 2001], the analysis of that paper shows that prime implicates satisfy all of the above requirements. This makes prime implicates in principle a good candidate compilation language. The empirical analysis can be used to evaluate whether this also holds in practice. A possible objection might be that the set of prime implicates is often very large, making this not a very suitable choice. However, some of the compilation method that are used have been shown to be able to cope with very large sets of prime implicates.

7.5 Experimentation: literature results

A limited number of empirical analyses have been reported in the literature. Of these, [Kautz and Selman, 1994] evaluates knowledge compilation of planning problems. They only evaluate unit bounds (i.e., the target compilation language consists of unit clauses), but show that such a restricted target language can already lead to substantial savings in computation time. However, the queries Kautz and Selman are trying to speed up answering time for are significantly different from ours. They are interested in speeding up answers to queries for *specific* planning problems (e.g., 'Do all plans for this problem involve a particular action?'). Such queries are only reasonable for a planning scenario with an initial state and goal formula.

In our experimental setup we use the same fixed-varying pattern used in [Liberatore, 1998]. Each planning instance consists of a domain theory, an initial state, and a goal formula. The domain theory is the fixed part and the initial state together with the goal formula is the variable part. Our motivation for this split is the need to solve many problem instances in the same world in many cases. Hence, we compile planning problems in order to speed up the construction of plans for *different* initial states and goal formulas.

In contrast with [Kautz and Selman, 1994], the information about initial state and goal formula is removed before performing knowledge compilation. It turns out that this leads to an enormous (and somewhat surprising) increase in computational complexity. This is shown in more detail in the next section.

This difference in experimental setup means that it is not possible to use the same compilation technique as [Kautz and Selman, 1994]. Removing the information about initial state and goal formula from the planning problems used in the empirical analysis resulted in the removal of all unit clauses from the theory. Using the technique from [Kautz and Selman, 1994] would lead to empty unit bounds and therefore useless compilations. From the results of [Schrag and Crawford, 1996b] follows that this is not only true for our benchmarks, but probably also for most other realistic planning problems.

When the initial state and goal formula is removed from the planning problem the result is a theory which is easily satisfied. To see this, take an arbitrary initial state and let the goal formula be satisfied in the initial state. Then a plan of length zero will satisfy the theory. Hence, considering the phase transition in random 3SAT [Mitchell *et al.*, 1992] it follows that the fixed part of the planning problem lies in a non-critical region. It was observed in [Schrag and Crawford, 1996b] that an approach that uses small prime implicates (e.g., unit clauses) to approximate satisfiable formulas is unlikely to formulate a non-empty approximation outside of the critical region.

There is a second reason why the algorithm used in [Kautz and Selman, 1994] cannot be used in the empirical analysis of this chapter. [Kautz and Selman, 1994] describes an adapted version of the algorithms in [Selman and Kautz, 1991]. In the experimental analysis of [Kautz and Selman, 1994] unit LUBs are computed in 5 minutes for 75 variable theories, to one hour for 200 variable theories. Reproducing these results actually led to different results. Closer inspection of the code, which was made available to by Kautz and Selman, revealed that an "encoding trick" was used. Generating all prime implicates is in general very hard. However, [Kautz and Selman, 1994] was only interested in a small subset of the prime implicates (namely prime implicates of length one (unit clauses)). Because this set is small, a simple generate and test algorithm sufficed. However, this optimization only works for small subsets of the prime implicates, but not in general.

7.6 Experimentation: our results

This section reports the experimental results with knowledge compilation applied to planning problems using the approach described in Section 7.2. Within this approach, several choices have to be made that can have an influence on the performance. These choices are *the problem domain*, the *encoding* used to represent the problem, and *the knowledge compilation method* used to compute the compilation. These choices are the major parameters in the empirical analysis. The following three subsections investigates the influence of the parameters on the knowledge compilation of planning problems.

7.6.1 Influence of different encodings

The first problem to be compiled was an instance from the Logistics domain, shown in Figure 7.2. For reasons that will become clear later, the *most trivial element* was deliberately chosen from the benchmark collection.

As stated before, the goal of knowledge compilation is to reduce the complexity of online reasoning by pre-processing a part of the problem in an off-line phase. In principle, the cost of this pre-processing stage is not important, since this step is performed off-line, and its costs can be amortized over solving many problem instances. However, the costs of this off-line stage must of course remain within humanly practical bounds. It will be shown that this is, rather surprisingly, not the case for even this trivial problem (remember that this problem can be solved with any reasonable planner almost instantaneously). In order to make this point, the compilation times are reported instead of problem-solving times. (Given the compilation, a SAT planner can solve the problem rather fast).

Encoding Information				Knowledge Compilation Information		
Medic Encoding	Nr. of Vars	Nr. of Clauses	Clauses var. part	KC Time fixed+var	KC Time fixed part	Factor of Increase
CBSE	29	165	6	0.6s	3024s	5040
EBSE	29	210	6	1.1s	9493s	8630
ERPE	38	117	6	0.6s	50s	83
CRSE	41	213	6	1.2s	-	-
EOSE	50	237	6	1.5s	624s	416
COSE	53	258	6	4.0s	-	-
ESSE	59	207	6	1.3s	370s	285
CSSE	62	345	6	4.3s	-	-
CCSE	170	381	6	8.8s	-	-
CFST	240	559	-	12.2s	-	-
ECSE	295	473	-	14.9s	-	-
EFST	365	786	-	38.3s	-	-

Figure 7.6: First results: Fifth Column shows the computation time to perform knowledge compilation on the fixed *and* variable part. The sixth column reports the knowledge compilation time for the fixed part only. The -'s in the time columns indicated that these did not return results after many hours of computation.

The experiments were started by generating twelve different encodings of the Logistics problem mentioned in Figure 7.2. These encodings are based on different switch settings used by the Medic Planner. The notation described in Section 7.3.2 is used for these switch settings.

The results of the first experiments are reported in Figure 7.6. The first column shows the encoding used. The second, third and fourth column gives some information about the encoding. The fifth and sixth column report the time needed to perform knowledge compilation. In the fifth column, the fixed and the variable part are used in the compilation whereas in the sixth column only the fixed part is used. (Note that the fifth column in Figure 7.6 resembles the experimental setup in [Kautz and Selman, 1994] discussed in Section 7.5.) Finally the last column shows the increase in computation time when the fifth and sixth column are compared. Several observations can be made from Figure 7.6, which are summarized below.

Influence of different encodings on knowledge compilation time: Looking at the fifth column (or the sixth) shows that there is a large difference in computation time var the various encodings. The experiment shows that in column five there is a difference of a factor of 60 in computation time between the results of the most efficient encoding CBSE and the most inefficient encoding EFST. Looking at the sixth column, where only the fixed part is used instead of the fixed and variable part, shows that the variation in computation time is even greater (including encodings that did not even yield an answer).

Not all encodings are equally useful for knowledge compilation: Before performing knowledge compilation, a SAT encoding is generated by giving the fixed *and* variable part to the Medic Planner. The SAT encoding is then split into a fixed and variable part. This splitting depends somewhat on the readability and size of the variable part and is therefore easier done for some encodings than others. In particular it was not obvious how to separate the fixed and variable parts under the EFST encoding. The splitting problem is more a problem of the used Medic Planner then a problem of knowledge compilation.

Removing variable part increases knowledge compilation time: As already mentioned in the previous section removing the variable part from a planning problem increases the computational complexity when performing knowledge compilation. This phenomenon can be seen in Figure 7.6 when the fifth and sixth column are compared. As can be seen, for some encodings computation time can increase by a factor of 8000! This illustrates that an alternative experimental setting as used in [Kautz and Selman, 1994] can dramatically change the applicability of knowledge compilation techniques to solve planning problems.

Increase of knowledge compilation time is unpredictable: It is very hard to predict the increase in computational complexity when the variable part is removed from a planning problem. Figure 7.6 shows that the CBSE and ERPE encodings take about the same time when knowledge compilation is performed on the fixed *and* variable part. However, when the variable part is removed, the computation time on the CBSE encoding increases with a factor of about 5000 while the ERPE encoding only increases with a factor of about 80. This behaviour of knowledge compilation techniques on planning problems will make it very hard to choose appropriate strategies for applying such techniques.

7.6.1.1 Domain specific knowledge

In [Kautz and Selman, 1998] it has been shown that domain dependent knowledge or heuristics can be added to a problem representation in order to decrease the computation time when solving the problem *on-line*. The same technique was applied and results show (Figure 7.7) that it can also be used to decrease the computation time when compiling the problem *off-line*.[ii]

The domain specific knowledge that was used consisted of just two rules: (1) An object is at only one location, and (2) Don't return a package to a location. The results in Figure 7.7 only report the gain when using both heuristics, but for these two rules the gain was incremental. Note that the first rule falls in the category 'state invariant' and the second rule in the category 'optimality heuristic' discussed in Section 6.3.2.2.

[ii]The compilation times reported in Figure 7.9, 7.7 and 7.10 do not include the variable part.

	Without domain knowledge		With domain knowledge		
Encoding	Time	Nr. of PI	Cl. added	Time	Nr. of PI
CBSE	3024s	124556	20	190s	29697
EBSE	9493s	124556	20	639s	29697
ERPE	50s	20809	20	8s	9255

Figure 7.7: Influence of adding domain specific knowledge.

Besides using automatically generated encodings, we were also interested in how much could be gained with domain specific hand-tailored encodings. One such encoding for the logistics domain is the State Based Encoding (SBE) from [Kautz and Selman, 1996].

The term ‘state-based’ was used in [Kautz and Selman, 1996] to emphasize the use of axioms that assert what it means for each individual state to be valid, and to give a secondary role to the axioms describing operators.

In the Logistics domain the state axioms include assertions that (1) each package can only be in a single unit (i.e., trucks and planes), (2) each packet or unit can only be at a single location, and (3) each packet is either at some location or in a unit, but not both at the same time.

The axioms for the state transitions can be formalized by using the schemas in Figure 7.8 that relate the `at` and `in` predicates.

```
at(obj,loc,i) ⇒                      in(obj,unit,i) ⇒
   at(obj,loc,i+1) ∨                    in(obj,unit,i+1) ∨
   ∃x ∈ truck ∪ airplane.               ∃x ∈ Location.
      in(obj,x,i+1) ∧                      at(obj,x,i+1) ∧
      at(x,loc,i+1) ∧                      at(unit,x,i+1) ∧
      at(x,loc,i+1)                        at(unit,x,i)
```

Figure 7.8: Schemas for generating the axioms that describe the state transitions.

The left schema asserts that if a package is at a location, it either remains at that location or goes into some truck or plane that is parked at that location. The right schema asserts that if a package is in a truck or plane, it either remains in that unit or becomes at the location where the unit is parked. Besides these two schemas, no additional axioms are needed for the operators that lead to state changes (e.g., `drive-truck`).

A solution to a state-based encoding of a planning problem results in a sequence of states and not in a sequence of actions. However, in the Logistics domain, the actions can easily be derived from the sequence of states.

The results of compilation with the state-based encoding are shown in Figure 7.9, which shows that this encoding improves compilation time even further.

The results presented here confirm the insights in the literature that choosing the right

Encoding	Nr. of Vars.	Nr. of Clauses	KC Time	Nr. of PI
SBE	20	85	1.0s	2870

Figure 7.9: Influence of hand-tailored encoding.

encoding for a specific domain is very important. Methodological support for this choice is therefore required.

7.6.1.2 Domain specific problems

The planning as satisfiability has a drawback not mentioned earlier in the context of knowledge compilation. To find an optimal plan, SAT planners use iterative deepening on the length of the plan in the propositional encoding. Longer plans contain larger points in time. Therefore, longer plans contain more points in time. Because every letter in the propositional encoding corresponds to one state at a certain point in time, longer plans need more letters for their encoding. From this, it follows that SAT planners *continually extend the alphabet* at each iteration. This scheme is not possible in knowledge compilation because the compilation will only be performed *once*, ahead of planning, with a *fixed* alphabet. It follows that we need to choose the alphabet large enough such that it can represent the optimal plan.

This is problematic because the optimal length of the plan is not known beforehand. If, on the one hand we choose an alphabet which is too small we may not be able to retrieve any plan from the compilation. If, on the other hand we choose an alphabet which is too large, our results (Figure 7.10) show that the cost for knowledge compilation may become too much because of an exponential increase. Even for knowledge compilation, which is performed off-line, such an exponential increase in the costs will quickly make this approach unfeasible for realistic domains.

Encoding	Max. length of plan	Nr. of Vars.	Nr. of Clauses	KC Time
SBE	3	20	85	1.0s
SBE	4	25	103	7.3s
SBE	5	30	127	30.0s
SBE	6	35	151	360.9s

Figure 7.10: Influence of increasing the length of the plan.

One might think this problem can be avoided by estimating an upper bound for the length of the plan and use this to determine the alphabet. At least two reasons indicate otherwise. First known complexity results tell us that determining the length of the plan is as hard as finding a plan [Cadoli, 1993]. Secondly, the bounds must be independent from the variable part. Thus the bound will be a loose one and Figure 7.10 shows that such a bound has a very high computational prize. For example, in the blocksworld this general bound is as high as $2n$ (n the number of blocks). This bound cannot be made any tighter (since problem instances with this minimal solution size exists), but is at the same time much to high for many other problem instances.

Problems as the one mentioned here are somewhat domain specific and make it very hard to choose the right knowledge compilation techniques for a specific domain.

7.6.1.3 Scalability

As we mentioned earlier, all results reported in previous sections were obtained by performing experiments on the most trivial benchmark problem. Although this already lead to severe problems, this becomes worse when we consider the scalability of the used techniques.

Small changes in a problem description can already lead to drastic increases in computation time. We extended the used benchmark problem by adding one location and one truck.

Whereas the problem in Figure 7.2 could be compiled within a second, the new problem already needed *30 minutes* for the compilation. For larger problems, we were not even able to get any results within several days of computation.

Because the problems used in our experiments are nowhere near the size of a real-world problem, this seriously casts a doubt on the usefulness of knowledge compilation without methodological support.

This is all the more surprising since we are using a recent implementation of knowledge compilation techniques, which in some cases could efficiently deal with up to 10^{60} prime implicates [Simon and delVal, 2001].

Although generating the set of prime implicates has been tried before in many areas (e.g., diagnostic reasoning), without much success due to the large number of prime implicates, we were surprised that the modern techniques employed in our experiments could not cope with even small extensions of the trivial input problem from Figure 7.2.

7.6.2 Influence of different domains

The various aspects that we will consider in this section are the hardness of solving an instance of the domain, symmetries that may occur in the domain, and subtheories written in a language which can be solved in polynomial time.

7.6.2.1 The hardness of instances

When considering the impact of the domain on knowledge compilation one of the questions that may rise is the difficulty to solve various instances of a planning problem by a SAT solver. If the instances are easily solved it would indicate that the problem is unsuitable for knowledge compilation, because the cost for knowledge compilation would have to be amortized over too many queries.

To measure the hardness of various instances we used the SAT solver zChaff to solve them, which is one of the fastest SAT solvers at the time of writing.

	Monkey-Test2			Hanoi-Strips4		
Enc.	# Vars	# Clauses	zChaff	# Vars	# Clauses	zChaff
CBSE	274	19671	21.62	489	48238	35.47
CCSE	1640	11191	1.99	1935	16039	49.94
CFST	1750	9078	0.42	2025	16444	227.89
COSE	418	19743	37.01	654	48658	51.18
CRSE	670	19680	1.92	1419	48253	1218.00
CSSE	490	20112	135.32	639	48643	44.10
EBSE	274	5460	0.86	489	17413	122.50
ECSE	4051	8292	0.03	3488	13452	49.98
EFST	4161	11030	0.14	3578	14937	174.21
EOSE	409	13659	0.07	639	44128	121.51
ERPE	661	5991	0.47	1404	18988	427.77
ESSE	481	13659	0.05	624	43408	17.39

Figure 7.11: The hardness to solve an instance from the Monkey and Tower of Hanoi domain measured by zChaff.

Figure 7.11 shows some of the obtained results for an instance of the Monkey domain and one from the Tower of Hanoi domain. The first column shows the encoding used to represent the instance. After the first column follows the results for the two instances. Both are divided into three columns. The first and second columns give information about the size of the problem by showing the number of variables and number of clauses respectively. Note that these numbers are with respect to the fixed *and* variable part of the problem, because we are dealing with an instance. The third column gives the running time in seconds of zChaff to solve the problem.

7.6.2.2 Symmetric difficulties

The removal of the initial state from the encoding makes knowledge compilation much harder because it introduces symmetries. This phenomenon can be seen in Figure 7.12, which shows a part of a planning problem from the logistics domain. The part consists of two locations A and B, both in which a truck is present. However, in the encoding of the fixed part we only encode the existence of the trucks and not their location, because this information is considered variable. Hence, the situation in Figure 7.12 can be instantiated in two ways. Both the left truck is called truck1 and the right truck truck2 or vice versa. Only one of these instantiations is necessary for finding all possible plans while the other instantiation can be safely omitted.

When compiling a theory all these semantically equivalent subproblems are included in the compiled theory. Hence, the compilation process "solves" many equivalent subproblems. This problem does not occur when solving a problem instance, because many variables are already given a value.

It depends on the planning domain how problematic these symmetric subproblems can be. For example, in the Logistics domain the symmetric subproblems are quite problematic because their number increases rapidly when the number of trucks and/or locations increases. However, in the Monkey domain this problem does not exist, because we only have one object of each type.

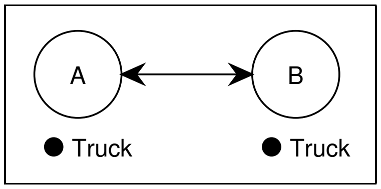


Figure 7.12: Symmetry in the Logistics domain.

Using the technique from Section 7.6.1.1, i.e., adding domain specific knowledge, some of these semantically equivalent subproblems can already be removed from the compilation process. For example, in the Logistics domain some trucks could already be assigned to cities. This is reasonable, as transport within a city is impossible when no truck is present. However, not all symmetries can be reduced this way. For example, in the Logistics domain the symmetries mentioned above for trucks also happen for packages for which nothing may be known about their starting locations.

Reducing symmetries by adding domain information can lead to an enormous decrease in encoding size. For example, we used the domain in Figure 7.12 where A and B now represent *cities*, both containing two locations. The encoding needed to represent plans with a maximum length of six steps. Using the state-based encoding this resulted in an encoding with 1581 clauses and 116 variables. When we added assertions to restrict the trucks to a city, the state-based encoding was reduced to an encoding with 479 clauses and 88 variables.

The reduction in encoding size is even more significant when the size of the problem increases. Adding a location to both cities A and B in the example shown in Figure 7.12 leads to a reduction of the state-based encoding with 8418 clauses and 158 variables to an encoding with 834 clauses and 116 variables.

7.6.2.3 Subtheories

A typical condition in a planning problem is for example "A truck can only be in a single location". Translating such a condition into a propositional formula results in a Horn formula. Because Horn theories can be solved in time linear in the length of the theory combined with the query [Dowling and Gallier, 1984] the question rises how much of a planning problem in propositional logic is written in an easy format.

We therefore computed the number of Horn formulas in the fixed part of our benchmarks. In Figure 7.13 the number of Horn formulas are given for a planning problem from the Monkey domain. The first three columns give information about the encoding used and the size of the fixed part of the planning problem. The fourth column gives the number of Horn formulas in the fixed part. The fifth column gives the percentage of Horn formulas in the fixed part with respect to the total number of clauses in the fixed part.

Medic Encoding	Fixed Part Nr of Variables	Fixed Part Nr of Clauses	Fixed Part Nr of Horn Clauses	Percentage of Horn Clauses
CBSE	274	19647	441	2.24
CFST	1750	6795	6462	95.01
COSE	418	19719	19647	99.63
CRSE	670	19656	19647	99.95
CSSE	490	20088	19647	97.80
EBSE	274	5436	1215	22.35
EOSE	409	11115	10467	94.17
ERPE	661	5967	5679	95.17
ESSE	481	13095	12447	95.05

Figure 7.13: The number of clauses of the Horn subtheory of the Monkey-Test2 problem under various encodings.

Figure 7.13 shows that the problem from the Monkey domain comprises of a remarkably high percentage of Horn clauses. Almost every encoding results in a theory with 95% or more Horn clauses. The only exceptions are the bitwise encodings.

However, a higher percentage in Figure 7.13 does not imply that any instance can be solved in less time using that encoding than any other encoding. We can easily see this by looking up the solution times in Figure 7.11. For example, the `CRSE` encoding has a percentage of 99.95 of Horn clauses, which is higher than the 94.17 percentage of the `EOSE` encoding. However, the `EOSE` encoding can be solved in 0.07 seconds which is smaller than the 1.92 seconds for the `CRSE` encoding. Clearly, other factors like number of variables or number of clauses are also important.

The results shown in Figure 7.13 do suggest that it might be profitable to split the theory into two parts, the Horn part, which consists of all the Horn clauses, and the non Horn part, which consists of the remainder. The high percentages suggest that it might be profitable to use the technique of theory prime implicates (Section 5.3.3) with respect to the set of all Horn clauses, because the larger the subset the smaller the compiled theory as was explained in Section 5.3.3. However, no *quantitative* measures are known for the size of the compiled theory and this is still part of our future work.

7.6.3 Influence of different knowledge compilation methods

Different methods have different performance profiles. In this section, we will look more closely at the different methods to see which method is more suitable for knowledge compilation of planning problems.

The first method we looked at was Kautz and Selmans LUB approximation (see Chapter 7.3.3 for a description of all knowledge compilation methods we used). We chose this method mainly for historical reasons, as it is one of the earliest knowledge compilation methods, and for reproducing results reported in the literature mentioned in Section 7.5. We did not use the method in any other experiment as it is easily outperformed by the other methods we used.

Besides comparing different knowledge compilation methods it is also possible to compare variations of the same method, because every knowledge compilation method comes with its own parameters which have to be set before it can be employed. Typical parameters are the target language and the order in which certain steps are performed. We will investigate the impact of some of these parameters on the effectiveness of knowledge compilation of planning problems.

7.6.3.1 Various orderings

Directional Resolution performs resolution steps, which depends on a given variable ordering. It is known that the performance of the algorithm can differ enormously under various variable orderings. However, to find the optimal variable ordering for a specific problem is NP-complete. To investigate the impact of various orderings we just started with two variable orderings that seemed reasonable. These orderings were:

Minimal Diversity Heuristic Ordering: For a variable the number of positive occurrences are counted as well as the number of all negative occurrences. These two numbers are multiplied and the result defines the order. A lower number indicates that less resolution steps will have to be performed for the related variable and should therefore be processed earlier than variables with a higher number.

Temporal Ordering: The encoding contains variables which refer to a certain point in time (e.g., variable x could represent "truck A is at location B at the point in time C"). Variables that refer to lower points in time are processed earlier than other variables. (Note that this resembles forward reasoning from the initial state.)

The influence of the orderings is shown in Figure 7.14 for the Monkey domain and in Figure 7.15 for the Tower of Hanoi domain. In both figures, the first column shows the encoding used. Thereafter follow two columns, one for each variable ordering used. Both are divided into two columns. The first column shows the time in seconds to perform the knowledge compilation. The second column shows the number of clauses in the compiled theory. A '-' in a column indicates that either the compilation process needed too much time or that the compiled theory contained more than a million clauses.

Figures 7.14 and 7.15 confirm that the performance of Directional Resolution is dependent on the chosen variable ordering as the results for both orderings differ. It also follows from Figures 7.14 and 7.15 that in this case there is no best ordering. In some cases the temporal ordering performs better, in other cases the minimal diversity heuristic ordering performs better. However, remarkably the orderings perform better with exactly the same encodings in both domains.

	Temporal Ord.		Min. div. heuristic Ord.	
Encoding	KC Time	Nr. Clauses	KC Time	Nr. Clauses
CBSE	187	185748	-	-
COSE	2260	621401	-	-
CRSE	1897	153305	507	54331
CSSE	-	-	-	-
EBSE	137	139282	-	-
EOSE	109	14550	109	24813
ERPE	182	14694	210	9586
ESSE	154	19701	177	37250

Figure 7.14: Overview of knowledge compilation time and size of the compilation for the Monkey planning problem under various encodings and two different variable orderings.

	Temporal Ord.		Min. div. heuristic Ord.	
Encoding	KC Time	Nr. Clauses	KC Time	Nr. Clauses
CBSE	-	-	-	-
COSE	-	-	-	-
CRSE	-	-	2474	143691
CSSE	-	-	-	-
EBSE	-	-	-	-
EOSE	313	16332	-	-
ERPE	1816	51702	1235	19306
ESSE	333	19455	-	-

Figure 7.15: Overview of knowledge compilation time and size of the compilation for the Hanoi planning problem with four rings under various encodings and two different variable orderings.

7.6.3.2 Exact versus approximate knowledge compilation

Until now, we have only considered exact knowledge compilation, i.e., the original theory was compiled into a logically equivalent theory. However, it would be interesting to see if computation time could be traded for approximate results. Basically there are two options to make knowledge compilation approximate (discussed in more detail in Section 5.4):

Language: The target language may be less expressive than the language used for the representation of the problem.

Algorithm: The algorithm can be interrupted to produce a partial compilation (i.e., anytime compilation).

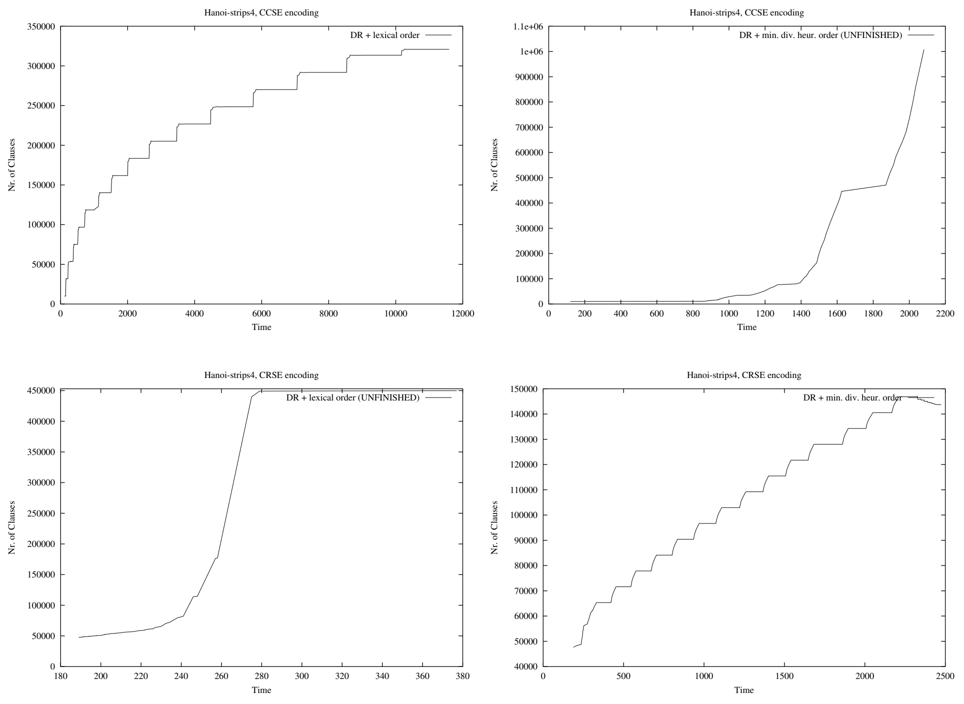


Figure 7.16: Performance profiles of Directional Resolution on the Hanoi four ring planning problem under various encodings.

Suppose that we use the language approximation technique and approximate the original theory Σ by a theory Σ'. Solving a planning problem with Σ' means finding a model of $\Sigma' \cup I \cup G$ where I and G are respectively the initial state and goal formula of the problem under consideration. Of course, when we find a solution we would also like this to be a solution with respect to the original theory. Hence, any model for $\Sigma' \cup I \cup G$ should also be a model for $\Sigma \cup I \cup G$. It follows that Σ' has to be a *lower bound* of Σ (i.e., $\Sigma' \models \Sigma$).

However, planning as satisfiability is an iterative process. In each step in this process the SAT planner is asked to answer the question "is there a plan of length n?" where n is set to one in the initial step and increased by one after each negative answer. Suppose that we approximate the original theory Σ by a theory Σ'. If we want the compilation to satisfy that whenever there is no model for $\Sigma' \cup I \cup G$ it also holds that there is no model for $\Sigma \cup I \cup G$ it follows that Σ' is an *upper bound* for the original theory Σ (i.e., $\Sigma \models \Sigma'$).

Obviously, Σ' cannot be a lower bound and an upper bound at the same time unless Σ' is logically equivalent to Σ. This means that if we want to approximate the theory, we need *two* approximations (i.e., a lower bound and an upper bound) or we can restrict ourselves to the use of a lower bound and loose the optimality of the found plan (i.e., the approximate

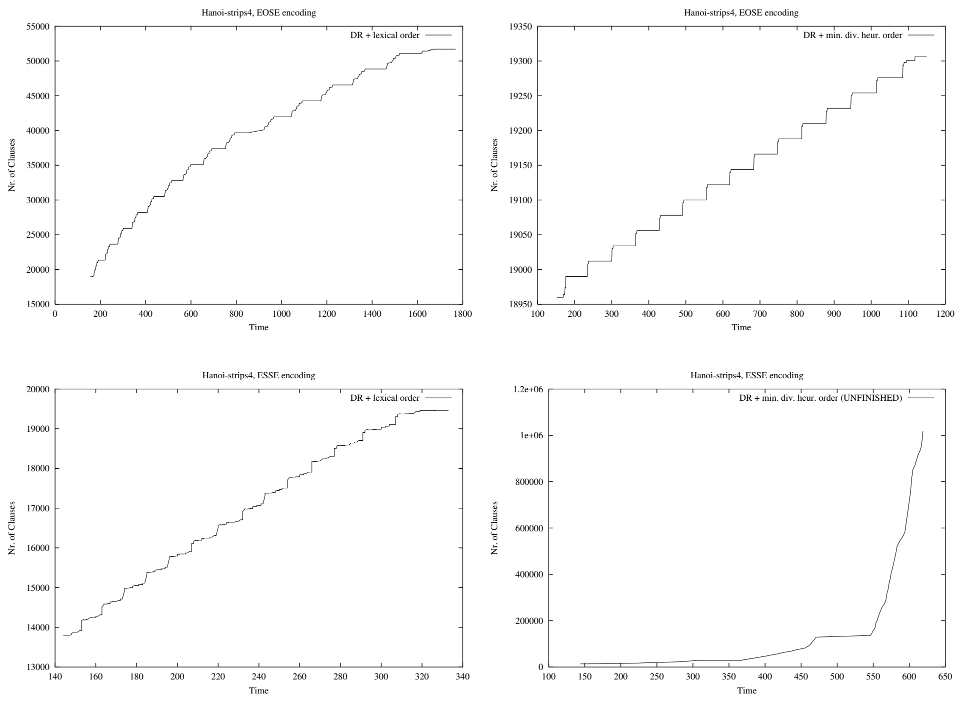


Figure 7.17: Performance profiles of Directional Resolution on the Hanoi four ring planning problem under various encodings.

solution found is guaranteed to be correct, but it might be that there is a more efficient, e.g., shorter, plan. However, the algorithms we have been considering cannot be used to generate a lower bound of a theory. We therefore postpone this step and leave it for future work.

However, the second option for approximation mentioned earlier is much easier to execute. Our first step was therefore to look at an anytime version of our knowledge compilation methods. Using Figure 7.15 to choose some interesting cases (i.e., where one of the variable orderings outperforms the other variable ordering), we ran Directional Resolution on them and generated their performance profiles (Figures 7.16 and 7.17).

In these performance profiles, we plot the size of the compiled theory against the time in seconds. Each row shows the performance profile on Directional Resolution on a specific encoding. The left subfigure in each row shows Directional Resolution with the temporal ordering and the right subfigure in each row shows Directional Resolution with the minimal diversity heuristic ordering. Note that in some cases only a partial performance profile is given as the compilation process was terminated before full completion (denoted by "Unfinished").

From the performance profiles, it can be seen that the growth of the compiled theory is enormous for all shown cases, even in those cases where the growth is linear or less than linear. It is well known that the main bottleneck of knowledge compilation is the enormous growth of the compiled theory. However, the size of the compiled theory is not the most important property. What really matters is if the compilation solves the problem knowledge compilation addresses: has on-line reasoning become more efficient using the compiled theory?

In order to measure the effectiveness of the compilation we therefore need to compare the time to solve an instance before and after compilation. However, the size of the compiled theory might influence this result. A possible scenario is depicted in Figure 7.18 where the time to solve an instance of a planning problem is mapped against the time invested in the knowledge compilation process. First, the compilation improves the solution time, but after some time, a point is reached where the compiled theory grows too large and reading/processing the compiled theory therefore takes too much time. This results in an increase in solution time.

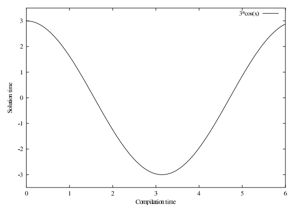


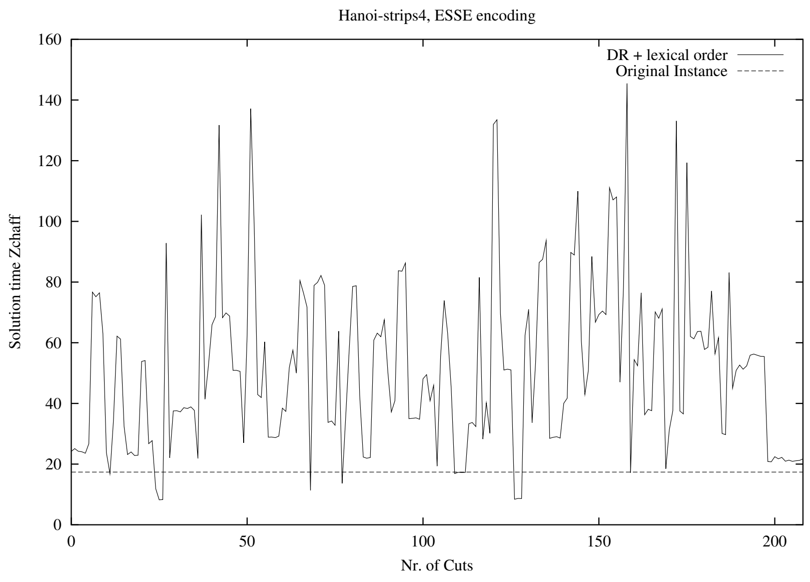
Figure 7.18: Possible scenario.

To get the optimal compilation with respect to the solution time we therefore generated new performance profiles in which we mapped the solution time of an instance against the steps taken by the algorithm. The results are shown in Figure 7.19. It should be clear that the behaviour shown in Figure 7.19 is nothing like the preferred performance profile of an anytime algorithm as it is non-monotonic and there is no significant improvement in solution time.

7.7 Conclusion

Knowledge compilation is a research direction for dealing with the computational intractability of propositional reasoning. The central idea behind knowledge compilation is that many problems can conceptually be split into a fixed part and a variable part. Planning, which is a notoriously hard problem, allows for such a split into a fixed part and a variable part and is therefore a prime candidate for the application of knowledge compilation techniques. Although knowledge compilation is theoretically appealing we argued that empirical studies are necessary to study the practical value of knowledge compilation. As this is a somewhat neglected area of current research we performed an empirical analysis of knowledge compilation on several planning problems. The empirical analysis was divided into three parts according to the three major parameters we identified that influence the efficiency of knowledge compilation: the problem domain, the encoding, and the knowledge compilation method.

The nature of the results of our empirical analysis makes our research by definition inconclusive as *all our obtained results are negative*. Such negative experimental results are



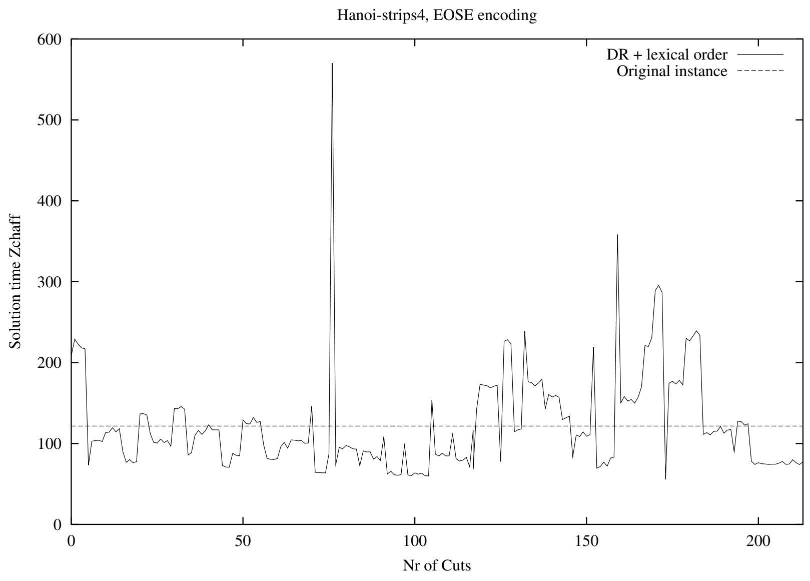


Figure 7.19: Performance profiles (with respect to solution time) of Directional Resolution on the Hanoi four ring planning problem under various encodings.

always inconclusive because it can always be argued that positive results could have been obtained if only we had used another experimental setup, another compilation method, another encoding method, etc. Nevertheless, to know what cannot be achieved with knowledge compilation gives us a greater understanding of its boundaries. More specifically, the following observations summarize our negative results:

1. Compilation time was high.
2. Compiled theory grew enormous.
3. Anytime profiles were non-monotonic.
4. Parameters were sensitive.
5. On-line computation time did not improve significantly.

As stated earlier, our research is still inconclusive. Therefore, no definitive choice can be made between the following two possible hypotheses we started with:

1. Knowledge compilation is not suitable for planning problems.
2. Knowledge compilation is suitabel for planning problems.

Nevertheless, some evidence can be given that supports the first conclusion. As we already discussed in Section 7.5, removing the initial state and goal formula from the planning problem leads to a theory which is easily satisfied. Hence, the fixed part of a planning problem belongs to the non-critical region. According to [Schrag and Crawford, 1996b] any knowledge compilation approach that uses small prime implicates to approximate satisfiable formulas is unlikely to formulate a non-empty approximation outside the critical region. It follows that our planning problems can only be compiled into a non-empty compilation by using *long* prime implicates. This has of course consequences for the size of the compilation and the speed of deducing facts from the compiled theory.

7.8 Future work

As not all possible choices for the parameters in the empirical analysis (e.g., knowledge compilation method, planning problem) have been explored, one can make other choices and use them for similar empirical analyses.

The presented analysis has focussed mainly on exact compilation of planning problems. A more promising direction for future research would be to look at approximate compilation techniques where some of the quality of the compiled theory is sacrificed in order to make the compilation more efficient. This will raise questions about how to evaluate these approximate compilations as the time to generate a solution is not the only quantity which needs measuring. For example, when applying approximate knowledge compilation to planning problems this may lead to a longer plan than the optimal one.

Another area worth considering is the use of other logics. When using propositional reasoning, the alphabet had to be chosen before we could perform knowledge compilation. Hence, for planning problems we already had to choose the maximal length of the plan for the solutions that could be found. This problem might be solved with for example first-order logic. Another logic may also lead to a more compact representation of a problem domain and therefore be a better candidate for knowledge compilation techniques.

These options for future research are, however, means towards the same goal of making knowledge compilation applicable for solving problems in planning. Two steps seem to be needed for making knowledge compilation suitable for applications in general. First, larger problems are needed that *are* compilable, i.e., knowledge compilation needs a set of *benchmarks*. These benchmarks can be used for individual research, but also for *competitions*. Second, knowledge compilation needs better guidelines. These guidelines should inform us which type of problem is (probably) a good candidate for knowledge compilation and which type of problem is (almost certainly) not.

7.9 Acknowledgments

We like to thank Laurent Simon and Alvaro del Val for making their implementation available as well as the productive correspondence we had through numerous e-mails.

We are also indebted to Henry Kautz and Bart Selman. They have provided us with the code they used in their experiments from '94, and they gave us insightful comments on earlier versions of our research. Furthermore, their work is one of the few in the AI literature, which is reported precisely enough that independent researchers can reproduce the results, as we have done in the early stages of our research.

Part III

Approximate Classification

Chapter 8

Classification

8.1 Introduction

A fundamental research issue in AI is how to reason in the presence of incorrect and/or incomplete information. AI tasks in particular use knowledge and observations which are often inherently incorrect and/or incomplete for some part. For example, in diagnosis, information may be incomplete, because obtaining it may be too costly or too dangerous for the patient. Also information may be incorrect, as patients may describe their symptoms incorrectly. Methods need therefore be developed that are robust enough to deal with data input that is not entirely correct or complete. This means that those kind of methods should generate the solutions that closely approximate the solutions that would be generated under ideal circumstances.

However, current approximation methods suffer from some limitations. First, some of the current methods use a numerical metric to measure the difference between optimal and approximate solutions. Such a metric is not applicable to many AI tasks as those tasks use logical inference to obtain solutions. Hence, there is no obvious way to tell us 'how far' an approximate solution is from the correct solution. Second, some of the methods are not general enough to apply them to tasks needed in new research areas. Third, for some of the methods that are general enough to apply them to more than one task, there is a gap between theory and practise because there are not enough case studies performed for the proposed methods. Therefore, it is unclear if those methods result in an approximation of the task which is useful enough for practice.

In this part of this thesis we present an analysis of the approximate entailment operator developed in [Schaerf and Cadoli, 1995] applied to the task of classification. There were several reasons that motivated this research. First, the proposed method of Cadoli and Schaerf allows us to approximate tasks that use logical entailment for obtaining solutions. Second, the method is general as it can be applied to any task that can be formulated in propositional logic and uses logical entailment for inferencing. Third, the method has some properties that are desirable for any general approximation method, which have not been evaluated beyond diagnosis in [vanHarmelen and tenTeije, 1995, tenTeije and vanHarmelen, 1996, tenTeije and vanHarmelen, 1997] by means of a quantitative and qualitative analysis. Fourth, classification seems to be a useful task to approximate as, for example, many tasks on the Semantic Web [Fensel *et al.*, 2003], which is currently rapid progressing as a research area, can be translated into terms of classification.

In general classification amounts to the following: given a number of classes defined in terms of their properties, determine for a specific object in which class it belongs. As some of the semantic information will be extracted from information sources that may be incorrect and/or incomplete (e.g., file system hierarchies, mail folder hierarchies, WWW-bookmark folders) one needs classification methods robust enough to be able to deal with these less than ideal information sources.

This part of the thesis is divided into three chapters. In this chapter we will address the question what is meant by classification, which forms of classification exist, and how to formalize classification. More specifically, Section 8.2 describes the conceptualization of classification given by [Jansen, 2003]. Section 8.3 gives several ways to formalize classification in propositional logic. Section 8.4 looks at some approximate forms of classification and formalizes them using standard propositional logic and the classical entailment operator. All concepts developed in this chapter will form the basis for the next chapters about approximate classification in which we study the effect of the approximate entailment operator developed in [Schaerf and Cadoli, 1995] applied to classification.

8.2 Characterizing classification

This section describes the conceptualization of classification given by [Jansen, 2003]. Classification can be characterized by defining three aspects: the goal of the task, the ontological commitments, and the classification criteria.

8.2.1 The goal

The goal in classification is to determine to which class a certain object belongs. The object is identified with a set of observations, which may be incomplete. Example classification tasks are bird classification, flower classification, and rock classification. Examples of object descriptions are the size of the bird, the colour of the flower, and the grain size of the rock.

8.2.2 Ontological commitments

The ontological commitments describe the assumptions on the representation of the domain to which classification may be applied. In [Jansen, 2003] six basic ontological types are defined, namely *attribute*, *object*, *value*, *class*, *feature*, and *observation*. An attribute is a quality which can be associated with a list of possible values. For example, the attribute *fur* can be associated with the list $\{white, black, brown\}$. The (finite) set of attributes is denoted with $\mathcal{A}$. An attribute-value pair (AV-pair) will be written as $\langle a, v\rangle$, or shorter a_v. An admissible AV-pair is called a feature. Objects that need to be classified are described by a finite number of AV-pairs. These AV-pairs are called observations. The set of observations for a particular object is called Obs.

In the conceptualization it is assumed that each attribute can only have one value at a time. For example, if there exists an attribute *fur* with possible values $\{white, black, brown\}$ then it can never be the case that an object description contains the AV-pairs $\langle fur, white\rangle$ and $\langle fur, black\rangle$ simultaneously.

Every attribute that has more than two values can be transformed into a number of binary attributes that only have $\{true, false\}$ as possible values. For example, the attribute *fur* with values $\{white, black, brown\}$ can be transformed into the three binary AV-pairs *fur-white*, *fur-black*, *fur-brown*.

Applying this transformation may lead to a conceptualization that does not enforce the assumption that each attribute can only have one value. After applying the transformation one has to additionally assume that at most one of the newly created binary AV-pairs can obtain the value *true*. In this way, each multi-valued attribute can be represented as a set of atomic propositions, which we will use when representing classification in propositional logic in Section 8.3.

8.2.3 Classification criteria

The classification criterion states what relationship should exist between a class and an object for the object to belong to the class. To determine whether an object, which is described by a number of AV-pairs, belongs to a certain class, the AV-pairs in the class must be compared to the observations. We assume that the same attributes occur in both the class and the observations. In this case, matching one AV-pair from a class with one from the observations is simply checking that they are syntactically identical. If the assumption does not hold, some mapping has to be performed from which it should be clear when two AV-pairs match.

Before formulating a classification criterion [Jansen, 2003] first considers all possible scenario's when an AV-pair from a class is compared to an AV-pair from the observations. This is done in two steps. First, the focus is limited to attributes. Second, values are included when needed.

8.2.3.1 The attribute level

[Jansen, 2003] identifies four possible scenario's when comparing an attribute in an AV-pair from the class and from the observations as not al attributes have to occur in a class or in the observations:

1. The attribute occurs in both the class and the observations.
2. The attribute does not occur in the class, but does occur in the observations.
3. The attribute occurs in the class, but not in the observations.
4. The attribute occurs neither in the class nor in the observations.

Next, [Jansen, 2003] considers these four options and also includes the value for the attributes when needed in the consideration. In the first case there are two possibilities:

1. The two values of the attribute do not match. In that case the two AV-pairs are said to be *inconsistent*.
2. The two values of the attribute match. Now the AV-pair in the observations is said to be *explained by* the matching AV-pair in the class. Similarly, the AV-pair in the class is said to *explain* (or is an *explanation for*) the matching AV-pair in the observations.

In the second case the attribute is present in the observation, but not in the class. There is no match possible, but this case does not lead to an inconsistency. The attribute in the observation is said to be *unexplained*.

In the third case, the attribute is present in the class, but not in the observations. Analogously to the second case, there is no match possible, but this case does not lead to an inconsistency. The attribute in the class is said *not to be an explanation* of the absent attribute in the observation.

The fourth case represents an empty statement. The attribute is neither observed nor mentioned in the class and does not lead to an inconsistency.

Summarizing the above mentioned cases, an attribute of a class c is said to be *consistent* with the observations o iff it does not occur in o or has the same value in o. Analogously, an attribute of an observation is consistent with a class c iff it does not occur in c or has the same value in c.

8.2.3.2 Classification criteria – definitions

The options on the attribute level can be used to formulate several classification criteria with respect to the goal of the classification process. In [Jansen, 2003] the following criteria (amongst others) are formulated:

Definition 8.2.3.1 (Weak classification) *A class c is a* weak *solution for a set of observations Obs iff all the AV-pairs in Obs are consistent with all the AV-pairs in c.*

Definition 8.2.3.2 (Strong classification) *A class c is a* strong *solution for a set of observations Obs iff all the AV-pairs in Obs are explained by some AV-pair in c.*

Definition 8.2.3.3 (Explanative classification) *A class c is an* explanative *solution for a set of observations Obs iff all the AV-pairs in c are explanations for some AV-pair in Obs.*

Definition 8.2.3.4 (Strong explanative classification) *A class c is a* strong explanative *solution for a set of observations Obs iff c is both a strong solution and an explanative solution for Obs.*

In case of class descriptions with multi-valued attributes we look at the subclasses corresponding to a specific combination of values for all the attributes. For example, let `bear` be the class that consists of the elements that have a `brown fur` or a `white fur`, and have a `large size`. The class `bear` can then be seen as the union of the two subclasses `white-bear` (white fur and large size) and `brown-bear` (brown fur and large size). A class with multi-valued attributes is then a weak solution if at least one of its subclasses corresponding to a value combination for its attributes is a weak solution. The other classification criteria are defined analogously for classes with multi-valued attributes.

8.2.3.3 Classification criteria – related facts

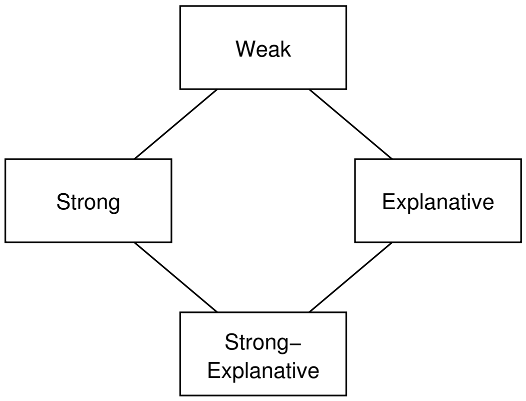


Figure 8.1: Lattice structure of various classification criteria in which an upward line means set inclusion.

The various classification criteria can be ordered into a lattice ([Jansen, 2003]). Every strong explanative solution is both a strong solution and an explanative solution. Furthermore, every strong solution or every explanative solution is also a weak solution. The lattice is shown in Figure 8.1.

Another way to look at strong- and explanative classification is the way they allow more or less observations than the ones mentioned in the class definitions. Every class can be identified with a set of AV-pairs. For a class to be a strong solution we are allowed to make *less* observations than the identified set of AV-pairs. For a class to be an explanative solution we are allowed to make *more* observations than the identified set of AV-pairs. This can be made more precise by defining the following sets of attributes ([Jansen, 2003]):

Definition 8.2.3.5 *Let c be a class, o a set of observations, and v some value. Then*

$$A_c = \{a \mid \langle a, v\rangle \in c\}, \quad A_o = \{a \mid \langle a, v\rangle \in o\}.$$

Now c is a strong solution iff $A_{Obs} \subseteq A_c$ and the values for the attributes occurring in A_{Obs} match. Analogously, c is an explanative solution iff $A_c \subseteq A_{Obs}$ and the values for the attributes occurring in A_c match.

8.3 Representation issues

There are several ways to represent classes as a logical formula: by necessary conditions, sufficient conditions, or a combination of both. We will investigate here how these different representations influences the formalization of the various classification criteria given in Section 8.2.3.2.

The alphabet we use consists of classes and observations (AV-pairs). The set of all classes will be denoted by $\mathcal{C}$. The set of all observations will be denoted by $\mathcal{O}$. The sets $\mathcal{C}$ and $\mathcal{O}$ are disjoint. Classes are usually denoted by c or d, whereas sets of classes are denoted by C or D. Observations are usually denoted by a, b, or o, whereas a set of observations is denoted by O. The object we need to classify can be identified with a set of AV-pairs, which is denoted by Obs. The representation of all class descriptions is called the domain theory and is denoted by DT.

Furthermore, we make a number of assumptions. We assume that $DT \cup Obs$ is consistent as well as $DT \cup \{c\}$ for any class $c \in \mathcal{C}$. We also assume $Obs \neq \varnothing$.

8.3.1 Classes defined by necessary conditions

One representation one can use for representing class descriptions is by stating them in terms of *necessary conditions* (e.g., [Wielinga *et al.*, 1998]). For example

$$c \rightarrow (a_1 \vee \ldots \vee a_n) \wedge \ldots \wedge (b_1 \vee \ldots \vee b_m).^{\text{i}}$$

A class is represented by a proposition c, which implies its features. Features are here represented by an atomic proposition, denoting the attribute, and an index, denoting the value. For example

$$bear \rightarrow (fur = white \vee fur = brown) \wedge (size = large).$$

Furthermore, we will assume that whenever $c \rightarrow a_1 \vee \ldots \vee a_n$ is part of the domain theory, each $a_1, \ldots, a_n$ contains the same attribute.

The representation that states class descriptions in terms of necessary conditions can be considered an alternative definition to the set theoretic conceptualization described in Section 8.2. All results given in [Jansen, 2003] for the set theoretic conceptualization can also be given for the representation of class descriptions by necessary conditions.

In Section 8.2.2 we assumed that an attribute can only have one value at a time. In case multi-valued attributes are transformed into a number of binary attributes we need to add additional rules to the domain theory to enforce this assumption. This can be done by adding rules for each feature of the following form ([Jansen *et al.*, 2000]):

$$a_1 \rightarrow (\neg a_2 \wedge \ldots \wedge \neg a_n).$$

[i]Note that $c \rightarrow a \wedge b$ is equivalent to $(c \rightarrow a) \wedge (c \rightarrow b)$ and equivalent to $\neg c \vee (a \wedge b)$. We will use these equivalent representations without further notice.

Under this representation the candidate solutions S with respect to some classification criteria can be formulated using logical entailment. We will first consider the case where class descriptions are given without disjunctions.

Definition 8.3.1.1 *Let DT be a domain theory containing class descriptions with necessary conditions in which disjunctions are not allowed. Then the classification criteria can be formalized as follows:*

Weak classification:

$$S_W = \{c \mid DT \cup \{c\} \cup Obs \nvdash \bot\}.$$

Strong classification:

$$S_S = \{c \mid DT \cup \{c\} \vdash Obs\} \cap S_W.$$

Explanative classification: *Let o represent an AV-pair, then*

$$S_E = \{c \mid \{o \mid DT \cup \{c\} \vdash o\} \subseteq Obs\} \cap S_W.$$

These definitions can also be found in the literature (e.g., [Jansen *et al.*, 2000]). Usually the definitions for strong- and explanative classification are simplified because of the following facts:

Lemma 8.3.1.2 *Every class c satisfying $DT \cup \{c\} \vdash Obs$ is a member of S_W.*

Proof Suppose $c \notin S_W$. Then $DT \cup \{c\} \cup Obs \vdash \bot$. As $DT \cup Obs$ (and $DT \cup \{c\}$) is consistent it follows there must be a rule of the form $c \rightarrow \neg o$ with $o \in Obs$. Therefore, $DT \cup \{c\} \nvdash o$ because this would contradict the assumption that $DT \cup \{c\}$ is consistent. Hence, $c \notin \{c \mid DT \cup \{c\} \vdash Obs\}$. □

Lemma 8.3.1.3 *Every class c satisfying $\{o \mid DT \cup \{c\} \vdash o\} \subseteq Obs$ is a member of S_W.*

Proof Suppose $c \notin S_W$. Then $DT \cup \{c\} \cup Obs \vdash \bot$. As $DT \cup Obs$ (and $DT \cup \{c\}$) is consistent it follows there must be a rule of the form $c \rightarrow \neg o$ with $o \in Obs$. As Obs is consistent we have $\neg o \notin Obs$ and $DT \cup \{c\} \vdash \neg o$. Hence, for c it does not hold that $\{o \mid DT \cup \{c\} \vdash o\} \subseteq Obs$. □

Hence, these results show that the intersection with S_W can be left out. Nevertheless, we will keep them in our definitions as it might influence results when we approximate the entailment operator that also occurs in S_W.

Of these three formula's only the one for weak classification can also be used in case of class descriptions that may contain disjunctions. For example, when the domain theory consists of the class description $c \rightarrow (a_1 \vee a_2) \wedge b$ and $\{a_1, b\}$ are observed then c should be given as a solution when using either strong- or explanative classification. However, with the definitions given above c is not part of the solution set as $\{c \rightarrow (a_1 \vee a_2) \wedge b, c\}$ does not entail a_1 (or a_2). Hence, the formalizations given above for strong- and explanative classification are only useful for class descriptions without disjunctions.

8.3.2 Classes defined by sufficient conditions

Another option is to represent classes by *sufficient* conditions, i.e.,

$$c \leftarrow (a_1 \vee \ldots \vee a_n) \wedge \ldots \wedge (b_1 \vee \ldots \vee b_m).$$

Obviously, this influences the logical representation of the various classification criteria. Each class c can be identified with a number of AV-pairs. In the following we denote this set of AV-pairs with obs_c.

Definition 8.3.2.1 *Let DT be a domain theory containing class descriptions with sufficient conditions in which disjunctions are not allowed. Then the classification criteria can be formalized as follows:*[ii]

Weak classification:

$$S_W = \{c \mid Obs \cup obs_c \not\vdash \bot\}.$$

Strong classification:

$$S_S = \{c \mid Obs \subseteq obs_c\} \cap S_W.$$

Explanative classification:

$$S_E = \{c \mid DT \cup Obs \vdash c\} \cap S_W.$$

Note that the definition of explanative classification can be simplified because of the following fact:

Lemma 8.3.2.2 *Each class c satisfying $DT \cup Obs \vdash c$ is a member of S_W.*

Proof A class c can only be logically entailed from $DT \cup Obs$ if $obs_c \subseteq Obs$. As Obs is assumed to be consistent it follows that c is a member of S_W. □

As the set obs_c can not be obtained through logical inference from the domain theory DT, the observations Obs, the class c, or any combination of them, any definition containing obs_c can not be formalized using propositional logic and some entailment relation. Therefore, only explanative classification can be used as a candidate for approximation when using the approximate entailment relation of Cadoli and Schaerf.

8.3.3 Classes defined by necessary- and sufficient conditions

A third representation uses a combination of necessary and sufficient conditions. In this case classes can be defined to be equivalent to a set of observations:

$$c \leftrightarrow (a_1 \vee \ldots \vee a_n) \wedge \ldots \wedge (b_1 \vee \ldots \vee b_m).$$

This representation will not be described in any more detail.

[ii]Note that we use the same names (S_W, S_S, and S_E) for the logical formalizations of the classification criteria as with class descriptions with necessary conditions. In the forthcoming sections it will be clear from the context which formalization we use.

8.4 Approximate classification

The classification criteria defined in Section 8.2.3.2 impose constraints on *all* attributes that occur in a class description and in the observations. It is possible to relax these constraints. Either by relaxing the presence of an attribute or by allowing inconsistencies. In Section 8.1 we motivated that situations occur in which a relaxation of a classification criterion is wanted. Classical classification may be to rigid to deal with incorrect/incomplete observations and/or domain knowledge. In the following sections we discuss into more detail how to formalize classification with missing attributes (Section 8.4.1), inconsistent attributes (Section 8.4.2), and a combination of missing and inconsistent attributes (Section 8.4.3).

Approximating classification by allowing missing or inconsistent attributes has also been discussed in [Jansen, 2003]. However, [Jansen, 2003] formulates approximate classification in terms of the concepts introduced in Section 8.2. Those concepts were given in a set-theoretic framework whereas we will use propositional logic and logical entailment for our formalization. Not only are we interested in formalizing approximate classification, we are also interested in analyzing the effects of the application of the approximate entailment operators of Cadoli and Schaerf to classification. We therefore formalize approximate classification that allows for missing or inconsistent attributes in the logical representations discussed in Section 8.3. This will allow us to compare the approximations we formalize in this section in which we use standard propositional logic and the classical entailment operator to the formalizations we obtain in Chapter 9 in which we use standard propositional logic and the approximate entailment operator of Cadoli and Schaerf.

Besides the choice of a different representation there is another distinction we make when considering missing or inconsistent attributes when compared to the work of [Jansen, 2003]. The formalizations given in this chapter make a clear distinction from which description (i.e., either class, object, or both descriptions) an attribute is allowed to be missing. In [Jansen, 2003] only attributes from the object description are allowed to be missing when strong classification is used, whereas only attributes from the class description are allowed to be missing when explanative classification is used.

8.4.1 Dealing with missing attributes

Missing observations may for example occur when obtaining the value for the attribute would be too costly (e.g., medical information from a patient).

Let us first discuss what we mean with classification where some of the attributes are allowed to be missing. We use strong classification as an example. In strong classification each attribute that has obtained a value should be explained by some AV-pair in the class description (i.e., there is a matching for every AV-pair in the observations Obs with some AV-pair in the class). In case we allow missing attributes, a class c is a strong solution if for each attribute, the corresponding AV-pair in Obs matches with an AV-pair in c, or the attribute does not occur in the object description, the class description, or both, and is also allowed to be missing from the description.

One way to model missing attribute values is by assigning the value ‘unknown’ to the attribute [Stefik, 1993]. In this case we need to include all attributes in the class definitions and the object description (i.e., $A_{Obs} = A_c$).

Each class can now be represented by ordering the attributes and listing first those attributes that have a known value followed by those that have an unknown value. This is shown in Figure 8.2. If we use the same attribute ordering used for the representation of the class to represent the observations we can immediately check if the class satisfies some classification criterion. This is illustrated in Figure 8.2 for strong- and explanative classification.

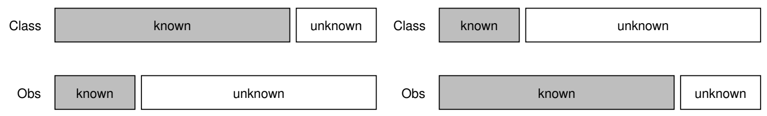


Figure 8.2: Illustration of a strong solution (left) and an explanative solution (right), which allows for missing attributes. The attributes of the observations and some class are abstractly represented as two rectangles. The dark rectangle contains the attributes with known values, the white rectangle contains the attributes with value unknown.

For example for strong classification the left subfigure in Figure 8.2 should hold, i.e., (1) for each attribute with value unknown in Obs the attribute either has value unknown in the class description, or it is allowed to be missing in the object description; (2) for each attribute with a value different from unknown in Obs the attribute either has the same value in the class, or has the value unknown in the class description and is allowed to be missing in the class description.

Note that with this formalization we are able to state *explicitly* which attributes are allowed to be missing. This is not possible for the classical classification criteria. For example, in strong classification it is impossible to allow missing attributes in the class description, whereas missing attributes in the object description are always accepted.

Further note that this formalization allows attributes missing in the class description and attributes missing in the object description to be treated *separately*. This separate treatment is different from the work of [Jansen, 2003]. In the case of strong classification, [Jansen, 2003] only explicitly states the attributes allowed to be missing in the object description. In the case of explanative classification, [Jansen, 2003] only explicitly states the attributes allowed to be missing in the class description.

In the following we denote with $A_{unknown}$ the set of attributes that are allowed to have the value unknown. Furthermore, we denote with $O_{unknown}$ the set $\{\langle a, unknown\rangle \mid a \in A_{unknown}\}$. We will use the shorthand a_u to denote the AV-pair $\langle a, unknown\rangle$. Strong classification that allows for missing attributes can now be formalized as follows:

Definition 8.4.1.1 (Strong classification with missing attributes) *Let DT be a domain theory containing class descriptions by necessary conditions in which disjunctions are not allowed. Then the solution set $S_S^{unknown}$ (i.e., strong classification in which certain* *attributes are allowed to be missing in either the class description or object description) can be modeled as follows:*

Missing attributes in the object description:

$$\{c \mid \forall a_v \in Obs : DT \cup \{c\} \vdash a_v \text{ or } a_v \in O_{unknown}\} \cap S_W =$$

$$\{c \mid DT \cup \{c\} \vdash Obs \setminus O_{unknown}\} \cap S_W.$$

Missing attributes in the class description:

$$\{c \mid \forall a_v \in Obs : DT \cup \{c\} \vdash a_v \text{ or } (a_u \in O_{unknown} \text{ and } DT \cup \{c\} \vdash a_u)\} \cap S_W.$$

Missing attributes in the class- or object description:

$$\{c \mid \forall a_v \in Obs : DT \cup \{c\} \vdash a_v \text{ or } a_v \in O_{unknown} \text{ or}$$

$$(a_u \in O_{unknown} \text{ and } DT \cup \{c\} \vdash a_u)\} \cap S_W.$$

The first formula formalizes strong classification in which certain attributes in the object description are allowed to be missing. This is similar to classical strong classification, but instead of allowing *all* attributes to be missing, now only those attributes explicitly mentioned in $A_{unknown}$ are allowed to be missing. In case we have $A_{unknown} = \mathcal{A}$, i.e., $A_{unknown}$ contains all attributes, then classical strong classification is the same as the solution set stated in the first formula. More precisely, the formula states that for every AV-pair a_v in Obs the attribute a should either (1) match with the value occurring in the class description, or (2) the attribute has an unknown value and is allowed to be missing. Note that in the above definition we make use of the fact that an AV-pair a_v is a member of a class c iff $DT \cup \{c\} \vdash a_v$. Hence, (1) holds whenever $DT \cup \{c\} \vdash a_v$ holds. (There is also a matching when the attribute has an unknown value in both the class description and object description.) When their is no match part (2) should hold. Note that $a_v \in Obs$ and $a_v \in O_{unknown}$ formalizes that attribute a is allowed to be missing in the object description and implies that $a_v = a_u$, i.e., that the attribute a has an unknown value in the object description.

The second formula formalizes strong classification in which certain attributes in the class description are allowed to be missing. The formula states that for every AV-pair a_v in Obs the attribute a should either (1) match with the value occurring in the class description, or (2) the attribute has an unknown value *in the class description* and is allowed to be missing. Part (1) is exactly the same as in the formalization of strong classification with missing attributes in the object description. When their is no match part (2) should hold. This is formalized as $a_u \in O_{unknown} \wedge DT \cup \{c\} \vdash a_u$. The first conjunct states that attribute a is allowed to be missing (note the use of a_u instead of a_v), whereas the second conjunct states that the attribute a has the value unknown in the class description.

The third formula formalizes strong classification in which certain attributes are allowed to be missing in both the class description and object description. It combines the other two formalizations through a disjunction.

Analogously to the case of strong classification we can consider missing attributes for explanative classification. The formalization is as follows:

Definition 8.4.1.2 (Explanative classification with missing attributes) *Let DT be a domain theory containing class descriptions by necessary conditions in which disjunctions are not allowed. Then the solution set $S_E^{unknown}$ (i.e., explanative classification in which certain attributes are allowed to be missing in either the class description or object description) can be modeled as follows:*

Missing attributes in the object description:

$$\{c \mid \forall a_v \in \{a_v \mid DT \cup \{c\} \vdash a_v\} : a_v \in Obs \text{ or } (a_u \in Obs \cap O_{unknown})\} \cap S_W.$$

Missing attributes in the class description:

$$\{c \mid \forall a_v \in \{a_v \mid DT \cup \{c\} \vdash a_v\} \setminus O_{unknown} : a_v \in Obs\} \cap S_W =$$

$$\{c \mid \{o \mid DT \cup \{c\} \vdash o\} \setminus O_{unknown} \subseteq Obs\} \cap S_W.$$

Missing attributes in the class- or object description:

$$\{c \mid \forall a_v \in \{a_v \mid DT \cup \{c\} \vdash a_v\} \setminus O_{unknown} : a_v \in Obs \text{ or } (a_u \in Obs \cap O_{unknown})\} \cap S_W.$$

The first formula formalizes explanative classification in which certain attributes in the object description are allowed to be missing. The formula states that for every AV-pair a_v in the class description the attribute a should either (1) match with the value occurring in the object description, or (2) the attribute has an unknown value *in the object description* and is allowed to be missing. Part (1) holds whenever $DT \cup \{c\} \vdash a_v$ holds. (There is also a matching when the attribute has an unknown value in both the class description and object description.) When their is no match part (2) should hold. This is formalized as $a_u \in Obs$ and $a_u \in O_{unknown}$. The first part states the attribute a has an unknown value in Obs (note the use of a_u instead of a_v), and $a_u \in O_{unknown}$ states that the attribute is allowed to be missing.

The second formula formalizes explanative classification in which certain attributes in the class description are allowed to be missing. The formula states that for every AV-pair a_v in the class description, which is not allowed to be missing, there should be a match with the corresponding attribute in Obs, i.e., a_v should be a member of Obs. No restrictions are imposed on any attribute a that has the value unknown in the class description and is also allowed to be missing. This is formalized by taking the set difference of all AV-pairs occurring in the class c, i.e., $\{o \mid DT \cup \{c\} \vdash o\}$, with $O_{unknown}$.

The third formula formalizes explanative classification in which certain attributes are allowed to be missing in both the class description and object description by combining the other two formalizations.

8.4.2 Dealing with inconsistent attributes

Besides allowing missing attributes it is also possible to allow certain attributes to be inconsistent (i.e., having a different value). It is not realistic to assume that every value obtained by observing our environment is equal to the value actually present in our environment. For example, this may happen because the measuring device used is too imprecise. Because of these incorrect observations, the object may not be classified in some classes in which it should be classified when all observations were correct. Dealing with this problem can be done by weakening the classification conditions used in the classification criteria by allowing inconsistencies for some attributes.

We use the same representation used in the previous section, i.e., every attribute is included in the class descriptions and the object description and some of these attributes may have the value unknown. Intuitively, every attribute which is allowed to be inconsistent should also be allowed to be missing. Furthermore, we allow inconsistencies in both the class description and the object description. In the case of an inconsistency, we only know that the two values compared are different. Therefore, we will not be able to distinguish if the inconsistency occurred because of errors in the class description or in the object description.

In the following we denote with $A_{incorrect}$ the set of attributes that are allowed to have an inconsistent value. Furthermore, we denote with $O_{incorrect}$ the set $\{\langle a, v\rangle \mid a \in A_{incorrect}\}$. The various classification criteria that allow inconsistent attributes can now be formalized as follows:

Definition 8.4.2.1 *Let DT be a domain theory containing class descriptions by necessary conditions in which disjunctions are not allowed. Then the various classification criteria can be weakened by allowing inconsistencies for observations in the set $O_{incorrect}$ as follows:*

Weak classification with inconsistent attributes:

$$S_W^{incorrect} = \{c \mid DT \cup \{c\} \cup (Obs \setminus O_{incorrect}) \not\vdash \bot\}.$$

Strong classification with inconsistent attributes:

$$S_S^{incorrect} = \{c \mid DT \cup \{c\} \vdash (Obs \setminus O_{incorrect})\} \cap S_W^{incorrect}.$$

Explanative classification with inconsistent attributes:

$$S_E^{incorrect} = \{c \mid \{o \mid DT \cup \{c\} \vdash o\} \setminus O_{incorrect} \subseteq Obs\} \cap S_W^{incorrect}.$$

The first formula formalizes weak classification in which the attributes occurring in $O_{incorrect}$ are allowed to be inconsistent. As weak classification only means checking the consistency of the observations with a class, this can be done by removing the attributes that are allowed to be inconsistent from the set Obs. Note that the formula can also be used in case class definitions are given that may contain disjunctions.

The second formula formalizes strong classification in which the attributes occurring in $O_{incorrect}$ are allowed to be inconsistent. If $O_{unknown} \subseteq O_{incorrect}$ it follows immediately that the set $S_S^{incorrect}$ formalized in Definition 8.4.2.1 includes the set $S_S^{unknown}$ formalized in Definition 8.4.1.1 of strong classification in which certain attributes are allowed to be missing in the object description (first formula). Obviously allowing an attribute to have a different value in the object description when compared to the attribute in the class description is equivalent to allowing an attribute to have a different value in the class description when compared to the attribute in the object description. Hence, the set $S_S^{incorrect}$ formalized in Definition 8.4.2.1 should also extend the second (and third) formulas in Definition 8.4.1.1 which formalizes strong classification in which certain attributes are allowed to be missing in the class description (or both object and class descriptions). The proof that strong classification with inconsistent attributes includes strong classification with missing attributes will be deferred to the next section.

The third formula formalizes explanative classification in which the attributes occurring in $O_{incorrect}$ are allowed to be inconsistent. Analogously to strong classification, from the fact that $O_{unknown} \subseteq O_{incorrect}$ it immediately follows that the set $S_E^{incorrect}$ formalized in Definition 8.4.1.1 includes the set $S_E^{unknown}$ formalized in Definition 8.4.1.2 of explanative classification in which certain attributes are allowed to be missing in the class description (second formula). The proof that explanative classification with inconsistent attributes also includes the other sets formalized in Definition 8.4.1.2 will be deferred to the next section.

8.4.3 Dealing with incorrect and missing attributes

Besides allowing only missing or only incorrect attributes it is also possible to allow some combination of missing and incorrect attributes. When $A_{incorrect}$ and $A_{unknown}$ are two disjoint sets one can easily take some disjunction or conjunction of the sets previously defined. However, when they are not disjoint the situation is different as an attribute cannot be missing and inconsistent at the same time.

Intuitively, if a class satisfies the observations according to some classification criterion in which an attribute is allowed to be missing, then this class should also be a solution according to the same classification criterion in which the same attribute is allowed to be inconsistent. Hence, solution sets with allowable missing attributes are included in solution sets with allowable inconsistent attributes (in both solution sets we take the same set of attributes that are allowed to be missing or inconsistent). This observation is mentioned in [Jansen, 2003] and confirmed by the following two lemma's for strong classification and explanative classification respectively:

Lemma 8.4.3.1 *If* $A_{unknown} = A_{incorrect}$ *then* $S_S^{unknown} \subseteq S_S^{incorrect}$.

Proof We will prove this for $S_S^{unknown} = \{c \mid \forall a_v \in Obs : DT \cup \{c\} \vdash a_v \vee a_v \in O_{unknown} \vee (a_u \in O_{unknown} \wedge DT \cup \{c\} \vdash a_u)\}$ as this set includes the other two solution sets defined in Definition 8.4.1.1. Suppose $c \notin S_S^{incorrect}$. Then there exists

$a_v \in Obs$ such that $DT \cup \{c\} \not\vdash a_v$ and $a_v \notin O_{incorrect}$. The latter implies that the attribute a does not occur in $A_{incorrect}$. Therefore, a does not occur in $A_{unknown}$ and a_v and a_u do not occur in $O_{unknown}$ and it follows that c is not a member of $S_S^{unknown}$. □

Lemma 8.4.3.2 *If $A_{unknown} = A_{incorrect}$ then $S_E^{unknown} \subseteq S_E^{incorrect}$.*

Proof We will prove this for $S_E^{unknown} = \{c \mid \forall a_v \in \{a_v \mid DT \cup \{c\} \vdash a_v\} \setminus O_{unknown} : a_v \in Obs \vee (a_u \in Obs \cap O_{unknown})\}$ as this set includes the other two solution sets defined in Definition 8.4.1.2. Suppose $c \notin S_E^{incorrect}$. Then there exists a_v such that $DT \cup \{c\} \vdash a_v$, $a_v \notin O_{incorrect}$, and $a_v \notin Obs$. As a does not occur in $A_{incorrect}$ it also does not occur in $A_{unknown}$. Therefore, a_v and a_u do not occur in $A_{unknown}$. Hence, $a_v \in \{a_v \mid DT \cup \{c\} \vdash a_v\} \setminus O_{unknown}$ and it follows that c is not a member of $S_E^{unknown}$.□

We have just proven that solution sets with allowable missing attributes are included in solution sets with allowable inconsistent attributes. The converse inclusion however, does not hold. Allowing an attribute to be missing does not mean that we allow the attribute to be inconsistent. When an attribute is *allowed* to be missing it does not mean that it actually *is* missing. The value of the attribute is allowed to be unknown, but when the value is known, the class is ruled out to be a solution whenever this value is inconsistent. The following example illustrates this.

Example 8.4.3.3 (Inconsistent attributes are not included in missing attributes) *Let $A = \{a, b\}$, $\mathcal{C} = \{c\}$, $\mathcal{O} = \{a_1, a_u, b_1, b_2, b_u\}$. Let the domain theory DT consist of the class description $c \rightarrow a_1 \wedge b_1$ together with all rules forcing each attribute to have exactly one value. Furthermore, let Obs $= \{a_1, b_2\}$ and $A_{unknown} = A_{incorrect} = \{b\}$. Then $O_{unknown} = \{b_u\}$ and $O_{incorrect} = \{b_1, b_2, b_u\}$. It can be verified with Definitions 8.4.1.1 and 8.4.2.1 that c is a member of $S_S^{incorrect}$ but not of $S_S^{unknown}$.*

This example confirming [Jansen, 2003] as it illustrates that classification in which missing attributes are allowed is different from classification in which inconsistent attributes are allowed.

8.5 Conclusion

In this chapter we have described classification, which will be used in the next chapter as a candidate task for approximation with the method developed in [Schaerf and Cadoli, 1995]. The study was performed because classification is a task that may benefit by being more robust for incorrect and incomplete class and object descriptions.

In this chapter we have described classification using the systematic approach of [Jansen, 2003]. The various ways in which classes and observations can be compared on the attribute level led to the formalization of several classification criteria giving several ways to classify an object into a class.

Thereafter, we showed how the conceptualization of [Jansen, 2003] in a set theoretic framework could be formalized into propositional logic. These encodings depended on the formalization of the class descriptions, which could either be formalized with necessary conditions, with sufficient conditions, or with a combination of both.

The last part of this chapter dealt with approximate forms of classification. We looked at classification that allowed for missing attributes and inconsistent attributes, which we consider to be useful approximations of classification. Missing and inconsistent attributes are also considered in [Jansen, 2003], but differs somewhat from the approach taken in this chapter. First, the approximations are formalized using propositional logic and the classical entailment operator instead of a set theoretic formalization. The propositional formalization allows us to compare the approximations in this chapter to the approximations that will be obtained in the next chapter using the method of Cadoli and Schaerf. Second, the formalizations in this chapter allows missing attributes in the object description and class descriptions to be treated separately, which is not considered by [Jansen, 2003].

This chapter should not be considered as an exhaustive description of the task. We have only presented those concepts needed in the next chapter where we will analyse the approximations obtained by applying the method of [Schaerf and Cadoli, 1995] to the formalizations in this chapter.

Chapter 9

Approximating classification

9.1 Introduction

In the previous chapter we have described what is meant by classification and gave several ways to formalize this task. As we motivated, classification, when used in its classic form, may be to rigid to deal with incorrect and/or incomplete class descriptions and object descriptions. In the previous chapter we already discussed in some detail various forms of approximate classification by explicitly stating which attributes are allowed to be missing or to be inconsistent. In this chapter we will also look at approximations of classification, but these approximations will be constructed by making use of a general approximation method.

The approximation method we will analyse in this chapter is the approximate entailment operator developed in [Schaerf and Cadoli, 1995]. By using this approximate entailment operator instead of the classical entailment operator in the formalizations given in the previous chapter, we will end up with an approximation of the various classification criteria. Although the method of Cadoli and Schaerf has some properties desirable for any general approximation method its unclear how useful these properties are in practice. For example, the method allows us to approximate a set S of solutions from below (and above) and the accuracy of the approximation can be improved in a stepwise fashion until we obtain S. If the method results in the sequence $\varnothing, \varnothing, \ldots, \varnothing, S$ then we are obviously no better of than computing S directly. In this chapter we will therefore analyse the effect of applying the method developed in [Schaerf and Cadoli, 1995] to classification, which we formalized in Chapter 8.

The rest of this chapter is structured as follows. First, in Section 9.2, we will give a summary of the approximate entailment operator developed by Cadoli and Schaerf. Thereafter, we will use this approximate entailment operator in the formalizations given in Chapter 8 and analyse the properties of the obtained approximations. We will start by analyzing the various classification criteria for class descriptions represented by necessary conditions. This will be done for weak classification in Section 9.3, for explanative classification in Section 9.4, and for strong classification in Section 9.5. The results obtained will be summarized in Section 9.6 and conclusions are given in Section 9.7.

9.2 Approximate entailment

This section gives a short overview of the approximate entailment by Cadoli and Schaerf [Schaerf and Cadoli, 1995] that allows for a weaker/stronger inference relation. Throughout this section we assume that there is an underlying finite language L used for building all the sentences. Symbols t and f are used for denoting special propositional letters, which are always mapped into 1 and 0, respectively. In the following we denote with S a subset of L.

Definition 9.2.0.4 (S-3-interpretation) *An S-3-interpretation of L is a truth assignment which maps every letter l of S and its negation $\neg l$ into opposite values. Moreover, it does not map both a letter l of $L \setminus S$ and its negation $\neg l$ into 0.*

Definition 9.2.0.5 (S-1-interpretation) *An S-1-interpretation of L is a truth assignment which maps every letter l of S and its negation $\neg l$ into opposite values. Moreover, it maps every letter l of $L \setminus S$ and its negation $\neg l$ into 0.*

The names given to the interpretations defined above can be explained as follows. For an S-1-interpretation there is *one* possible assignment for letters outside S, namely false for both x and $\neg x$. For an S-3-interpretation there are *three* possible assignments for letters outside S, namely the two classical assignments, plus true for both x and $\neg x$. (As a classical interpretation allows *two* possible assignments for letters, such an interpretation is sometimes referred to as a 2-interpretation.)

Satisfaction of a formula by an S-1- or S-3-interpretation is defined as follows. The formula is satisfied by an interpretation σ if σ evaluates the formula written in Negated Normal Form (NNF) into true using the standard rules for the connectives.

The notions of S-1- and S-3-entailment are now defined in the same way as classical entailment: A theory T S-1-entails a formula ϕ, denoted by $T \models_1^S \phi$, iff every S-1-interpretation that satisfies T also satisfies ϕ. S-3-entailment is defined analogously and denoted by $T \models_3^S \phi$.

Let $S, S' \subseteq L$, T a generic propositional CNF formula, and ϕ a generic propositional clause not containing both a letter l and its negation $\neg l$. We use the shorthand $\models_i^S \Rightarrow \models_i^{S'}$ denote $T \models_i^S \phi \Rightarrow T \models_i^{S'} \phi$. These definitions then lead to the following result [Schaerf and Cadoli, 1995]:

Theorem 9.2.0.6 (Approximate Entailment) *Let $S, S' \subseteq L$, such that $S \subseteq S'$, then*

$$\models_3^{\varnothing} \Rightarrow \models_3^S \Rightarrow \models_3^{S'} \Rightarrow \models_2 \Rightarrow \models_1^{S'} \Rightarrow \models_1^S \Rightarrow \models_1^{\varnothing} .$$

This theorem tells us that $\models_3^S$ is a sound but incomplete approximation of the classical entailment $\models_2$, whereas $\not\models_1^S$ is a sound but incomplete approximation of $\not\models_2$ (i.e., $\not\models_1^S \Rightarrow \not\models_1^{S'} \Rightarrow \not\models_2$). Furthermore, the theorem states that the accuracy of the approximations can be improved by increasing the parameter S until the approximations coincide with the classical entailment.

Theorem 9.2.0.6 holds even if T is a NNF formula and ϕ is a generic formula in CNF. This aspect has been analyzed in [Cadoli and Schaerf, 1995], which analyzes other normal forms for which the result holds.

We will give some other properties that hold for S-1- and S-3-entailment, which will be useful in the upcoming sections. The first lemma shows that a familiar property of $\models$ also holds for all approximations:

Lemma 9.2.0.7 (Monotonicity of $\models_i^S$)

$$If\, T \models_i^S \phi \;\; then \;\; T \wedge x \models_i^S \phi.$$

We will continue with some results, which show that S-i-entailment can be reduced to S-i-satisfiability. Before doing so we introduce the following definition:

Definition 9.2.0.8 *We denote with $letters(\gamma)$ the set $\{l \in L \mid l \; occurs \; in \; \gamma\} \cup \{l \in L \mid \neg l \; occurs \; in \; \gamma\}$.*

The next two theorems show that S-1- and S-3-entailment can be reduced to S-1- and S-3-satisfiability, respectively.

Theorem 9.2.0.9 (Reducing S-1-entailment to S-1-satisfiability) *Let γ be $\gamma_S \cup \gamma_{\overline{S}}$, where both letters($\gamma_S$) $\subseteq S$ and letters($\gamma_{\overline{S}}$) $\cap S = \varnothing$ hold. $T \models_1^S \gamma$ holds iff $T \cup \{\neg\gamma_S\}$ is not S-1-satisfiable.*

Theorem 9.2.0.10 (Reducing S-3-entailment to S-3-satisfiability) *Let letters(γ) $\subseteq S$ hold. $T \models_3^S \gamma$ holds iff $T \cup \{\neg\gamma\}$ is not S-3-satisfiable.*

Note that the condition $letters(\gamma) \subseteq S$ in Theorem 9.2.0.10 is not a restriction since [Schaerf and Cadoli, 1995] also prove that $T \models_3^S \gamma$ iff $T \models_3^{S \cup letters(\gamma)} \gamma$.

These results extend the well-known relation existing between classical entailment and satisfiability, namely $T \models \gamma$ iff $T \wedge \neg\gamma$ is unsatisfiable. The importance of such a result is that S-3-satisfiability can be tested in the following way:

1. replace by t all occurrences (both positive and negative) in T of letters which belong to $L \setminus S$, thus obtaining the formula $[T]_3^S$.
2. test standard (2-valued) satisfiability of $[T]_3^S$.

In a similar way S-1-satisfiability can be tested in the following way:

1. replace by f all occurrences (both positive and negative) in T of letters which belong to $L \setminus S$, thus obtaining the formula $[T]_1^S$.
2. test standard (2-valued) satisfiability of $[T]_1^S$.

Hence, considering the above tests we can clarify S-1- and S-3-satisfiability by the following syntactic operations. For a theory T in clausal form, T is S-1-satisfiable iff T is classically satisfiable after removing from every clause any literals with a letter outside S. When this results in an empty clause, the theory becomes the inconsistent theory $\perp$. Similarly, T is S-3-satisfiable iff T is classically satisfiable after removing every clause from the theory that contains a literal with a letter outside S. This may result in the empty theory $\top$. Because of the close correspondence between S-1-, S-3-satisfiability and these syntactic operations, we prefer to write $\vdash_i^S$ instead of $\models_i^S$ in the following sections.

So far, we have given some definitions, shown how these definitions approximate the classical entailment operator, and how the approximate entailment operators can be reduced to some form of satisfiability checking. We conclude this section by looking at the computational costs of the method.

Cadoli and Schaerf present an algorithm that can be used to compute the S-1-satisfiability and S-3-unsatisfiability of a generic formula. This algorithm runs in time exponential in $|S|$ and uses polynomial space. Furthermore, the algorithm can benefit from previous computations. More precisely, when $S' \supset S$, computing satisfiability with respect to S' can be done by using information gained in a previous step when the satisfiability was computed with respect to S. Hence, the method can be used to approximate the classicial entailment operator from two directions (by using $\not\vdash_1^S$ and $\vdash_3^S$ instead of the $\vdash$ operator) in a stepwise fashion and is not harder (and usually easier) to compute than the original problem.

9.3 Approximating weak classification

In this section we will investigate the influence of the approximate entailment relations of Cadoli and Schaerf on weak classification where the class descriptions are given by necessary conditions (Section 8.3.1). The approximations are obtained by selecting either $\vdash_1^S$ or $\vdash_3^S$ instead of the usual entailment and additionally choosing an appropriate set of propositional letters S. More precisely, we define

$$S_{W1}^S = \{c \mid DT \cup \{c\} \cup Obs \not\vdash_1^S \perp\},$$

$$S_{W3}^S = \{c \mid DT \cup \{c\} \cup Obs \not\vdash_3^S \perp\},$$

for some set S of propositional letters. The following result shows that these definitions can be used to approximate weak classification from two directions:

Lemma 9.3.0.11 *Let $S, S' \subseteq L$, such that $S \subseteq S'$. Then*

$$\varnothing = S_{W1}^{\varnothing} \subseteq S_{W1}^S \subseteq S_{W1}^{S'} \subseteq S_W \subseteq S_{W3}^{S'} \subseteq S_{W3}^S \subseteq S_{W3}^{\varnothing} = \mathcal{C}.$$

Proof This follows immediately from Theorem 9.2.0.6. □

We give some examples illustrating this lemma showing that S^S_{W1} and S^S_{W3} result in sub- and supersets of S_W. (Remember that for observations o_1 and o_2 of the same type we need to include rules in the domain theory of the form $o_1 \to \neg o_2$ to enforce the assumption that each attribute can only have one value at a time (Section 8.3.1).)

Example 9.3.0.12 ($\boldsymbol{S^S_{W1} \subsetneq S_W}$) *Let $DT = \{c \to o_1 \vee o_2, \neg o_1 \to o_2\}$ and $Obs = \{\neg o_1\}$ then $c \in S_W$, but $c \notin S^{\{o_1,c\}}_{W1}$.*

Example 9.3.0.13 ($\boldsymbol{S_W \subsetneq S^S_{W3}}$) *Let $DT = \{c \to o_1, \neg o_1 \to o_2\}$ and $Obs = \{o_2\}$ then $c \notin S_W$, but $c \in S^{\{o_2,c\}}_{W3}$.*

The rest of this section is structured as follows. In Section 9.3.1 we start by deriving some results that restrict the choice of the parameter S. Thereafter, in Section 9.3.2, we will derive results on how the approximations can be interpreted.

9.3.1 Restrictions on choosing S

Not every choice for the parameter S will lead to an useful approximation of weak classification. For example, taking the empty set for S will always result in S^S_{W1} being empty and S^S_{W3} being the set of all possible classes. In this section we will derive some relations between the parameter set S and the solution sets S^S_{W1} and S^S_{W3}.

9.3.1.1 Restrictions for S with respect to S^S_{W1}

In this section we will look at relations between the parameter S and the set of solutions S^S_{W1}. The first result on choosing S is given in the following lemma. It states that for a class c to be a solution it has to be included in the parameter S:

Lemma 9.3.1.1 *If $c \in S^S_{W1}$ then $c \in S$.*

Proof If a class $c \notin S$ then $DT \cup \{c\} \cup Obs$ will always evaluate into false with an S-1-assignment. Hence $DT \cup \{c\} \cup Obs \vdash^S_1 \bot$ and $c \notin S^S_{W1}$. □

Equivalently this result also tells us that we can restrict the set of candidate solutions by choosing the parameter S to be small. Any class c left out of S will not be part of the candidate solutions.

The following result relates the parameter S with the observations Obs:

Lemma 9.3.1.2 *If $S^S_{W1} \neq \varnothing$ then $Obs \subseteq S$.*

Proof Suppose there exists an observation $o \in Obs \setminus S$. Any S-1-assignment will map o into false. Therefore, $DT \cup \{c\} \cup Obs$ will be mapped into false, and $DT \cup \{c\} \cup Obs \vdash^S_1 \bot$. Hence, for every class c, c will not be a solution according to S^S_{W1}. □

The result above gives us a lower bound on the parameter S for obtaining non-trivial subsets of weak classification. Whereas the above two results restricted S in relation to either Obs or $\mathcal{C}$, the following result restricts S in relation to the domain theory DT:

Lemma 9.3.1.3 *If $c \in S^S_{W1}$ and $c \to a_1 \vee \cdots \vee a_n$ is an element of the domain theory DT then one of the a_i will be in S.*

Proof By definition of S^S_{W1} we have $DT \cup \{c\} \cup Obs \not\vdash^S_1 \perp$. Hence, there exists an S-1-assignment ϕ that maps $DT \cup \{c\} \cup Obs$ into true. Therefore, DT, and in particular $c \to a_1 \vee \cdots \vee a_n$, is mapped into true by ϕ. As $c \in S$ (Lemma 9.3.1.1), ϕ maps c into true and $\neg c$ into false. Therefore, one of the elements $a_1, \ldots, a_n$ has to be mapped into true and this element belongs to S, because all letters outside S are mapped to false. □

9.3.1.2 Restrictions for S with respect to S^S_{W3}

In this section we will look at relations between the parameter S and the set of solutions S^S_{W3}. We start by giving a lower bound on the parameter S:

Lemma 9.3.1.4 *If $c \notin S$ then $c \in S^S_{W3}$.*

Proof Suppose $c \notin S$, then the mapping that maps both c and $\neg c$ into true can easily be extended to an S-3-assignment that maps $DT \cup \{c\} \cup Obs$ into true. Hence, $DT \cup \{c\} \cup Obs \not\vdash^S_3 \perp$ and $c \in S^S_{W3}$. □

This result tells us that it is not useful to exclude any class c from the parameter set S as otherwise it will always be a solution. The only choices for S we need to consider are therefore those sets that includes the set of all classes $\mathcal{C}$.

The following result restricts S with respect to the observations Obs (under reasonable assumptions):

Lemma 9.3.1.5 *If $DT \cup \{c\} \not\vdash \perp$ and $Obs \cap S = \varnothing$ then $c \in S^S_{W3}$.*

Proof an S-3-assignment amounts to removing all clauses with letters outside S. Therefore, $DT \cup \{c\} \cup Obs \not\vdash^S_3 \perp$ iff $DT \cup \{c\} \not\vdash^S_3 \perp$. The latter follows immediately from Theorem 9.2.0.6 and $DT \cup \{c\} \not\vdash \perp$. □

This result tells us that we have to include some aspect of the object description in the parameter S, because else we will end up with all classes in the set of solutions (note that we assume $DT \cup \{c\} \not\vdash \perp$ for every $c \in \mathcal{C}$).

The following lemma restricts S with respect to the domain theory DT:

Lemma 9.3.1.6 *If $c \notin S^S_{W3}$ then there exists a clause $c \to a_1 \vee \cdots \vee a_n$ in DT with $\{a_1, \ldots, a_n\} \subseteq S$.*

Proof Given some class c, suppose that for every clause of the form $c \to a_1 \vee \cdots \vee a_n$ in DT there exists a_i such that $a_i \notin S$. Then every S-3-assignment amounts to removing every clause containing c from the domain theory DT. As we assume $DT \cup Obs$ to be consistent it follows that $DT \cup \{c\} \cup Obs \not\vdash_3^S \bot$. Hence, $c \in S_{W3}^S$. □

This lemma gives us a necessary condition for a class c not to be a member of S_{W3}^S. If one wants to determine if $c \notin S_W$ it is sufficient to find a parameter S such that $c \notin S_{W3}^S$ (Lemma 9.3.0.11). Any set S that does not contain a set $\{a_1, \ldots, a_n\}$ with $c \to a_1 \vee \ldots \vee a_n$ in DT can be ruled out for this purpose.

We finish this section by giving an example to illustrate the results obtained in this section.

Example 9.3.1.7 *Let $\mathcal{A} = \{a, b\}$ and each attribute can have two values. Let the domain theory DT be the set $\{c \to (a_1 \vee a_2) \wedge b_1, a_1 \to \neg a_2, b_1 \to \neg b_2\}$. Suppose Obs $= \{b_2\}$. As $\{c, c \to b_1, b_1 \to \neg b_2, b_2\}$ is inconsistent, c is classically not a weak solution. Now let us consider various choices for S and the effect this has on S_{W3}^S. By Lemma 9.3.1.4 if $c \notin S$ then $c \in S_{W3}^S$ and by Lemma 9.3.1.5 if $b_2 \notin S$ then $c \in S_{W3}^S$. For the other options let us therefore assume that $\{c, b_2\} \subseteq S$. By Lemma 9.3.1.6 either $\{a_1, a_2\}$ or $\{b_1\}$ should be included in S or else c will always be included in S_{W3}^S. It can be checked that when $S = \{c, b_2, a_1, a_2\}$ we have $c \in S_{W3}^S$ and that when $S = \{c, b_2, b_1\}$ we have $c \notin S_{W3}^S$. Note that in the first case the literal b_1 is excluded from S, whereas in the second case b_1 is included in S. The literal b_1 was the reason for rejecting c as a classic weak solution as this contradicted the observation b_2. This observation will be made more precise in Section 9.3.2.2.*

9.3.2 Interpreting S_{W1}^S and S_{W3}^S

In Section 9.3.1 we gave some results that related the parameter S to the solution sets S_{W1}^S and S_{W3}^S. In this section we continue this analysis, by analyzing the solution sets S_{W1}^S and S_{W3}^S in more detail. Where possible, we will give an interpretation of the solution sets in terms of the approximate classifications constructed in Chapter 8 where we modeled classification that allowed for missing and inconsistent attributes.

9.3.2.1 Interpreting S_{W1}^S

Lemma 9.3.1.1 showed us that a class c can be removed from the solution set of S_{W1}^S by removing c from the parameter S.

Lemma 9.3.2.1 *Let $D \subseteq \mathcal{C}$. Then $S_{W1}^{D \cup \mathcal{O}} \subseteq S_W \cap D$.*

Proof Let $c \in S_{W1}^{D \cup \mathcal{O}}$. Then by definition $DT \cup \{c\} \cup Obs \not\vdash_1^{D \cup \mathcal{O}} \bot$ and therefore by Theorem 9.2.0.6 $DT \cup \{c\} \cup Obs \not\vdash \bot$. Hence, $c \in S_W$. From Lemma 9.3.1.1 follows that $c \in D \cup \mathcal{O}$. Hence, $c \in D$. □

The reverse inclusion does not hold in general as the following example shows:

Example 9.3.2.2 ($\boldsymbol{S_{W1}^{D\cup\mathcal{O}} \subsetneq S_W \cap D}$) *Let $DT = \{c_1 \rightarrow o_1, c_2 \rightarrow \neg o_1\}$, $Obs = \{\neg o_1\}$, and $D = \{c_2\}$. As an S-1-assignment amounts to removing all literals from clauses that do not occur in S, $DT \cup \{c_2\} \cup Obs \not\vdash_1^{\{c_2\}\cup\mathcal{O}} \bot$ iff $\{o_1, c_2 \rightarrow \neg o_1, c_2, \neg o_1\} \not\vdash \bot$. As this does not hold we have $c_2 \notin S_{W1}^{D\cup\mathcal{O}}$. However, $DT \cup \{c_2\} \cup \{\neg o_1\}$ is consistent (i.e., mapping o_2 and c_1 into false, and mapping c_2 into true results in a model) and therefore $c_2 \in S_W \cap D$.*

This example shows that c_2 is not a member of $S_{W1}^{D\cup\mathcal{O}}$ because the inconsistent class c_1 is left out of the parameter S. This leads us to the following result:

Lemma 9.3.2.3 *If $c \notin S_W$ then $S_{W1}^{(\mathcal{C}\setminus\{c\})\cup\mathcal{O}} = \varnothing$.*

Proof Suppose there exists a class $d \in S_{W1}^{(\mathcal{C}\setminus\{c\})\cup\mathcal{O}}$. Then $d \neq c$ by Lemma 9.3.1.1. Furthermore, $DT \cup \{d\} \cup Obs \not\vdash_1^{(\mathcal{C}\setminus\{c\})\cup\mathcal{O}} \bot$. This holds iff $[DT \cup \{d\} \cup Obs]_1^{(\mathcal{C}\setminus\{c\})\cup\mathcal{O}} \not\vdash \bot$ iff $DT_c \cup DT_d \cup \{c\} \cup \{d\} \cup Obs \not\vdash \bot$ where $DT_c = \{c \rightarrow a_1 \vee \cdots \vee a_n \in DT\}$. However, this does not hold as we assumed $c \notin S_W$ from which follows $DT_c \cup \{c\} \cup Obs \vdash \bot$ and therefore by monotonicity $DT_c \cup DT_d \cup \{c\} \cup \{d\} \cup Obs \vdash \bot$. □

Note that for any two subsets A and B such that $A \subseteq B$ we have $S_{W1}^{A} \subseteq S_{W1}^{B}$. The previous result therefore tells us that we only have to consider those sets for the parameter S that include all inconsistent classes.

Let us now consider the effect of leaving out observations from the parameter S. Assume that $\mathcal{C} \subseteq S$ and $Obs \subseteq S$ (the latter is necessary for S to be non empty by Lemma 9.3.1.2). Now $c \notin S_{W1}^{S}$ holds by definition when $DT \cup \{c\} \cup Obs \vdash_1^S \bot$. The latter holds iff $[DT \cup \{c\} \cup Obs]_1^S \vdash \bot$, which by assumption holds iff $[DT]_1^S \cup \{c\} \cup Obs \vdash \bot$. This last formula holds for example when there exists a clause $c \rightarrow a_1 \vee \ldots \vee a_n$ in DT such that for some m between 1 and n it holds that $a_1, \ldots, a_m \in S$, $\neg a_1, \ldots, \neg a_m \in Obs$, and $a_{m+1}, \ldots, a_n \notin S$ as an S-1-entailment amounts to removing all literals with letters not in S.

The previous results show that applying S-1-entailment to weak classification leads to some complex form of approximate classificiation that is not easily characterized, either in terms of classification that allows for missing or inconsistent attributes or any other form. In the rest of our analysis we therefore restrict ourselves to the application of 3-entailment to the various classification criteria. We will show that applying S-3-entailment to the various classification criteria will lead to approximations that can be characterized in terms of the formalizations discussed in Chapter 8.

9.3.2.2 Interpreting $\boldsymbol{S_{W3}^{S}}$

In xample 9.3.1.7 we noted that a class c, which is not a weak solution classically, may be an approximate weak solution according to S_{W3}^{S} when the inconsistent literals are not in S.

We will make this more precise in this section. This will be done by relating S^S_{W3} to the approximation $S^{incorrect}_W$ constructed in Section 8.4.2, which formalizes weak classification in which the AV-pairs occurring in $O_{incorrect}$ are allowed to be inconsistent.

As we showed in Lemma 9.3.1.4, any reasonable choice for S should include the set of all classes $\mathcal{C}$. We therefore analyse only the effect of including certain observations in S. The following result shows that for $S = \mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect})$ we always obtain an upper bound of $S^{incorrect}_W$.

Lemma 9.3.2.4 $S^{incorrect}_W \subseteq S^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}_{W3}$.

Proof Let $c \in S^{incorrect}_W$. Then $DT \cup \{c\} \cup (Obs \setminus O_{incorrect}) \not\vdash \bot$ and therefore by Theorem 9.2.0.6 $DT \cup \{c\} \cup (Obs \setminus O_{incorrect}) \not\vdash_3^{\mathcal{C}\cup\mathcal{O}\setminus O_{incorrect}} \bot$. As $Obs \cap \mathcal{C} = \varnothing$, this is equivalent to saying $DT \cup \{c\} \cup (Obs \cap (\mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect}))) \not\vdash_3^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})} \bot$. Since, an S-3-assignment amounts to removing all clauses with letters outside S, the latter formula holds iff $DT \cup \{c\} \cup Obs \not\vdash_3^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})} \bot$. Hence, $c \in S^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}_{W3}$. $\square$

Note that by Lemma 9.3.0.11, for any set S that is included in $\mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect})$, S^S_{W3} will also be an upper bound of $S^{incorrect}_W$ (i.e., if $S \subseteq \mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect})$ then $S^{incorrect}_W \subseteq S^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}_{W3} \subseteq S^S_W$.). In fact, we will prove that $S^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}_{W3}$ is equal to $S^{incorrect}_W$ making $S^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}_{W3}$ trivially the closest upper bound one can find of $S^{incorrect}_W$. The converse inclusion is proven in the following lemma:

Lemma 9.3.2.5 $S^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}_{W3} \subseteq S^{incorrect}_W$.

Proof Suppose that $c \notin S^{incorrect}_W$. Then $DT \cup \{c\} \cup (Obs \setminus O_{incorrect})$ is inconsistent by definition. By assumption $DT \cup \{c\}$ is consistent and in each clause of DT only observations of the same type (or a class) occur. Therefore there is some attribute a and a subset $DT_a = \{c \rightarrow a_1 \vee \ldots \vee a_n \mid a_i \textit{ contains attribute } a\} \cup \{a_1 \rightarrow \neg a_2 \wedge \ldots \wedge \neg a_n \mid a_i \textit{ contains attribute } a\}$ of DT such that

$$DT_a \cup \{c\} \cup (Obs \setminus O_{incorrect}) \vdash \bot.$$

As $DT_a \cup \{c\}$ is consistent by monotonicity, the attribute a has to occur in $Obs \setminus O_{incorrect}$. Hence, attribute a occurs in $S = \mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect}) \supseteq (Obs \setminus O_{incorrect})$. More precisely, because of the way S was constructed, *every* AV-pair containing attribute a is a member of S. As an S-3-entailment only removes clauses with a letter outside S it follows that $DT_a \cup \{c\} \cup (Obs \setminus O_{incorrect})$ is a subset of $[DT \cup \{c\} \cup Obs]^S_3$. Hence $DT \cup \{c\} \cup Obs \vdash \bot$, and $c \notin S^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}_{W3}$. $\square$

As we showed that $\mathcal{C}$ should always be included in S and the results above characterize the effect of excluding observations from S, we obtain the main result of this section: an interpretation of S^S_{W3}.

Theorem 9.3.2.6 $S_{W3}^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})} = S_W^{incorrect}$.

Proof This follows from Lemma 9.3.2.4 and Lemma 9.3.2.5. □

This result gives us an interpretation of S_{W3}^S and also gives us a method for computing $S_W^{incorrect}$. The complexity of computing S_{W3}^S depends on the size of the parameter S (i.e., the algorithm given by Cadoli and Schaerf runs in time exponential in $|S|$). We noted before that the upper bound we found was the closest upper bound one could find of $S_W^{incorrect}$, because they were in fact the same sets. Hence, the complexity of computing $S_W^{incorrect}$ is at most exponential in $|S|$. However, $\mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect})$ is not necessarily the smallest parameter choice one could use to compute $S_W^{incorrect}$. For example, suppose one has a domain theory DT and observations Obs such that $S_W = \mathcal{C}$, i.e., all classes are solutions for the standard weak classification criterion. From $S_W \subseteq S_W^{incorrect}$ and $S_W \subseteq S_{W3}^S$ it follows that we also have $S_W^{incorrect} = S_{W3}^S = \mathcal{C}$ for *any* choice of the parameter S. In particular this will hold for $S = \varnothing$. Hence, the complexity of computing S_{W3}^S is at most exponential in the size of $\mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect})$ but may be less.

We finish this section by illustrating with an example that the assumption that in any clause of DT only observations of the same type occur, is necessary for proving the main result of this section, i.e., $S_{W3}^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})} = S_W^{incorrect}$.

Example 9.3.2.7 *Let $\mathcal{A} = \{a, b\}$ and each attribute can have two values. Furthermore, let $\mathcal{C} = \{c\}$ and $\mathcal{O} = \{a_1, a_2, b_1, b_2\}$. Suppose that $DT = \{c \to a_1, c \to \neg a_1 \vee \neg b_1, a_1 \to \neg a_2, b_1 \to \neg b_2\}$, $Obs = \{b_1\}$, and $A_{incorrect} = \{a\}$. Then $O_{incorrect} = \{a_1, a_2\}$. Note that $DT \cup \{c\}$ is consistent satisfying our assumptions. However $DT \cup \{c\} \cup Obs$ is inconsistent and therefore $c \notin S_W$. As $Obs \setminus O_{incorrect} = Obs$ it follows that $DT \cup \{c\} \cup Obs \setminus O_{incorrect}$ is also inconsistent and therefore $c \notin S_W^{incorrect}$. Let us now consider S_{W3}^S for the parameter choice $S = \mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect}) = \{c, b_1, b_2\}$. As an S-3-entailment amounts to removing every clause with letters outside S it follows that for $S = \{c, b_1, b_2\}$ this amounts to removing every clause that contains either a_1 or a_2. Hence, every rule containing c will be removed from DT as every such rule contains a_1 ($c \to a_1$ or $c \to \neg a_1 \vee b_1$). Hence, $[DT \cup \{c\} \cup Obs]_3^S$ is consistent and therefore $c \in S_{W3}^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}$. This illustrates that when the domain theory DT does not satisfy the assumption that in every clause of DT only observations of the same type (or a class) occur, the equality $S_W^{incorrect} = S_{W3}^{\mathcal{C}\cup(\mathcal{O}\setminus O_{incorrect})}$ will not hold in general.*

9.4 Approximating explanative classification

In this section we will investigate the influence of the approximate entailment relation of Cadoli and Schaerf on explanative classification where the class descriptions are given by necessary conditions (Section 8.3.1). The approximations are obtained by selecting either $\vdash_1^S$ or $\vdash_3^S$ instead of the usual entailment and additionally choosing an appropriate set of propositional letters S. As the entailment is also used in the definition of weak classification S_W, the entailment is used twice in the definition of explanative classification. More precisely, we get the following approximations

$$S^S_{E1} = \{c \mid \{o \mid DT \cup \{c\} \vdash^S_1 o\} \subseteq Obs\} \cap S^S_{W1},$$

$$S^S_{E3} = \{c \mid \{o \mid DT \cup \{c\} \vdash^S_3 o\} \subseteq Obs\} \cap S^S_{W3},$$

for some set S of propositional letters. The following result shows that these definitions can be used to approximate explanative classification from two directions:

Lemma 9.4.0.8 *Let $S, S' \subseteq L$, such that $S \subseteq S'$. Then*

$$\varnothing = S^{\varnothing}_{E1} \subseteq S^S_{E1} \subseteq S^{S'}_{E1} \subseteq S_E \subseteq S^{S'}_{E3} \subseteq S^S_{E3} \subseteq S^{\varnothing}_{E3} = \mathcal{C}.$$

Proof (We prove $S^S_{E1} \subseteq S_E \subseteq S^S_{E3}$. The rest is proved analogously or follows from the definitions of an S-1- or S-3-assignment.) Whenever $DT \cup \{c\} \vdash o$ holds, also $DT \cup \{c\} \vdash^S_1 o$

The rest of this section is as follows: we start by deriving some results that restrict the choice of the parameter S; thereafter we will derive results on how the approximations can be interpreted.

9.4.1 Restrictions on choosing S

Not every choice for the parameter S will lead to an useful approximation of explanative classification. For example, taking the empty set for S will always result in S^S_{E1} being empty and S^S_{E3} being the set of all possible classes. In this section we will derive some relations between the parameter set S and the solution set S^S_{E3}.

Before deriving any restrictions on S we will need the following result:

Lemma 9.4.1.1 *If $c \notin S$ then $\{o \mid DT \cup \{c\} \vdash^S_3 o\} = \varnothing$.*

Proof $DT \cup \{c\} \vdash^S_3 o$ holds iff $DT \cup \{c, \neg o\} \vdash^{S \cup \{o\}}_3 \perp$ holds. Let ϕ be the $(S \cup \{o\})$-3-assignment that maps both c and $\neg c$ into true, $\neg d$ into true for every $d \in \mathcal{C} \setminus \{c\}$, and $\{\neg o\}$ into true. Then $DT \cup \{c, \neg o\}$ is mapped into true by ϕ. Therefore, $\{o \mid DT \cup \{c\} \vdash^S_3 o\} = \varnothing$. □

The set $\{o \mid DT \cup \{c\} \vdash^S_3 o\}$ is part of a condition used in the definition of explanative classification. Lemma 9.4.1.1 states that whenever c is not a member of the parameter S the set $\{o \mid DT \cup \{c\} \vdash^S_3 o\}$ will be empty irrespective of any other variable in or not in S. Under this condition (i.e., $c \notin S$), we can simplify the definition of explanative classification which leads us to the following result:

Lemma 9.4.1.2 *If $c \notin S$ then $c \in S^S_{E3}$.*

Proof By Lemma 9.4.1.1 we have $\{o \mid DT \cup \{c\} \vdash_3^S o\} = \varnothing \subseteq Obs$. Hence, $c \in \{c \mid \{o \mid DT \cup \{c\} \vdash_3^S o\} \subseteq Obs\}$. In Lemma 9.3.1.4 we proved $c \in S_{W3}^S$, hence $c \in S_{E3}^S$. □

This result gives us a lower bound on reasonable choices for the parameter S. Lemma 9.4.1.2 states that it is not useful to exclude any class c from the parameter S as otherwise it will always be a solution. The only choices we need to consider are therefore those sets that include the set of all classes $\mathcal{C}$.

9.4.2 Interpreting S_{E3}^S

In this section we give a characterization of S_{E3}^S. In fact, we will prove that changing the entailment relation in explanative classification to S-3-entailment will not change the set of solutions when the parameter S includes the set of all classes $\mathcal{C}$. As we showed in Lemma 9.4.1.2 that it is not useful to exclude any class c from the parameter S, because this will add c to the solution set, this will characterize S_{E3}^S completely.

Before we are able to derive the characterization of S_{E3}^S we will need the following result, which states that the condition $\{o \mid DT \cup \{c\} \vdash o\}$ used in the definition of explanative classification does not change when the entailment operator is approximated by the S-3-entailment operator of Cadoli and Schaerf.

Lemma 9.4.2.1 *If the domain theory DT contains class descriptions without disjunctions and $\mathcal{C} \subseteq S$, then*

$$\{o \mid DT \cup \{c\} \vdash_3^S o\} = \{o \mid DT \cup \{c\} \vdash o\}.$$

Proof The set inclusion $\{o \mid DT \cup \{c\} \vdash_3^S o\} \subseteq \{o \mid DT \cup \{c\} \vdash o\}$ follows directly from Theorem 9.2.0.6. We will prove the set inclusion $\{o \mid DT \cup \{c\} \vdash o\} \subseteq \{o \mid DT \cup \{c\} \vdash_3^S o\}$. Suppose $a \in \{o \mid DT \cup \{c\} \vdash o\}$. Then there exists $c \rightarrow a$ in DT. Now $DT \cup \{c\} \vdash_3^S a$ iff $DT \cup \{c\} \cup \{\neg a\} \vdash_3^{S \cup a} \perp$ iff $[DT]_3^{S \cup a} \cup \{c\} \cup \{\neg a\} \vdash \perp$. As $c \rightarrow a$ is an element of $[DT]_3^{S \cup a}$ this holds and therefore $a \in \{o \mid DT \cup \{c\} \vdash_3^S o\}$. □

This lemma shows us that when $\mathcal{C} \subseteq S$, the parameter S has no effect on the condition $\{o \mid DT \cup \{c\} \vdash_3^S o\}$ used in explanative classification. As classically any explanative solution is also a weak solution (Lemma 8.3.1.3), any solution which is not a weak solution cannot be an explanative solution. This holds in particular for any $c \in S_{W3}^S \setminus S_W$. Hence, such a class c is not an element of $\{c \mid \{o \mid DT \cup \{c\} \vdash o\} \subseteq Obs\}$ and therefore by Lemma 9.4.2.1 not an element of $\{c \mid \{o \mid DT \cup \{c\} \vdash_3^S o\} \subseteq Obs\}$. This leads us to the following characterization of S_{E3}^S:

Theorem 9.4.2.2 *Let $D \subseteq \mathcal{C}$, and $O \subseteq \mathcal{O}$. If the domain theory DT has only class descriptions without disjunctions then*

$$S_{E3}^{(\mathcal{C} \setminus D) \cup O} = S_E \cup D.$$

Proof This follows from Lemmas 9.4.2.1 and 8.3.1.3. □

This theorem states that the usefulness of approximating explanative classification by using S-3-entailment is limited. There is no interpretation of S_{E3}^S possible in terms of explanative classification in which missing or inconsistent attributes are allowed as there is a one to one correspondence between classes not in S and classes in S_{E3}^S. More precisely, any sequence of sets $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_n$ used as parameter in S_{E3}^S to approximate explanative classification with increasing accuracy as S_i increases, leads to the sequence of solution sets $S_E \cup D_1 \supseteq S_E \cup D_2 \supseteq \cdots \supseteq S_E \cup D_n$, where $D_i = \mathcal{C} \setminus S_i$, approximating S_E in an obvious but uninformative way.

9.5 Approximating strong classification

In this section we will investigate the influence of the approximate entailment relation of Cadoli and Schaerf on strong classification where the class descriptions are given by necessary conditions (Section 8.3.1). The approximations are obtained by selecting either $\vdash_1^S$ or $\vdash_3^S$ instead of the usual entailment and additionally choosing an appropriate set of propositional letters S. As the entailment is also used in the definition of weak classification S_W, the entailment is used twice in the definition of strong classification. More precisely, we get the following approximations

$$S_{S1}^S = \{c \mid DT \cup \{c\} \vdash_1^S Obs\} \cap S_{W1}^S,$$

$$S_{S3}^S = \{c \mid DT \cup \{c\} \vdash_3^S Obs\} \cap S_{W3}^S,$$

for some set S of propositional letters.

Whenever $DT \cup \{c\} \vdash_3^S Obs$ then also $DT \cup \{c\} \vdash Obs$ by Theorem 9.2.0.6. Thus $\{c \mid DT \cup \{c\} \vdash_3^S Obs\} \subseteq \{c \mid DT \cup \{c\} \vdash Obs\}$. However, in Lemma 9.3.0.11 we proved $S_W \subseteq S_{W3}^S$. Hence, following this reasoning, S_{S3}^S is in general neither a lower- or upper bound of strong classification. Analogously, it can be shown that S_{S1}^S is also in general neither a lower- or upper bound of strong classification.

Although it cannot be determined at this point if the approximations of strong classification are a lower- or upper bound of strong classification, its unclear how they should be used to approximate strong classification. Nevertheless, we will analyse their properties. (Actually, S_{S3}^S is an upper bound of strong classification, but the reasoning is more complex and will therefore be deferred. The fact that S_{S3}^S is an upper bound will become clear after we characterize S_{S3}^S for every possible choice for the parameter S in Section 9.5.2.)

The rest of this section is as follows: we start by deriving some results that restrict the choice of the parameter S; thereafter we will derive results on how the approximations can be interpreted.

9.5.1 Restrictions on choosing S

Not every choice for the parameter S will lead to an useful approximation of strong classification. For example, taking the empty set for S will always result in S_{S1}^{S} being empty and S_{S3}^{S} being the set of all possible classes. In this section we will derive some relations between the parameter set S and the solution set S_{S3}^{S}.

Lemma 9.5.1.1 *If $c \notin S$ then $c \notin S_{S3}^{S}$.*

Proof This follows directly from $\{o \mid DT \cup \{c\} \cup \vdash_3^S o\} = \varnothing$ (Lemma 9.4.1.1) and the assumption that $Obs \neq \varnothing$ as this implies $c \notin \{c \mid Obs \subseteq \{o \mid DT \cup \{c\} \cup \vdash_3^S o\}\}$ or equivalently $c \notin \{c \mid DT \cup \{c\} \cup \vdash_3^S Obs\}$. Hence, $c \notin S_{S3}^{S}$. □

This result tells us that we can restrict the set of candidate solutions by choosing the parameter S to be small. Any class c left out of S will not be part of the candidate solutions.

9.5.2 Interpreting S_{S3}^{S}

We start this section by analyzing the effect of leaving out classes from the parameter S. Thereafter, we will analyse the effect of leaving out observations from the parameter S, which will lead to a characterization of S_{S3}^{S}.

In Lemma 9.5.1.1 we already proved that leaving out a class c of the parameter S will result in c not being a member of the solution set S_{S3}^{S}. We will prove that this is precisely the resulting effect when classes are left out of the parameter S. More precisely, leaving out classes from the parameter S is exactly S_{S3}^{S} (with all classes in S) restricted to a subset of the class hierarchy. The proof of this result will be divided into two parts. First we prove that the strong classification S_{S3}^{S} restricted to some part of the class hierarchy D (i.e., $S_{S3}^{\mathcal{C}\cup O} \cap D$) is an upper bound of the strong classification $S_{S3}^{D\cup O}$ (i.e., strong classification in which certain classes are removed from the parameter S.).

Lemma 9.5.2.1 *Let $D \subseteq \mathcal{C}$, and $O \subseteq \mathcal{O}$. Then $S_{S3}^{D\cup O} \subseteq S_{S3}^{\mathcal{C}\cup O} \cap D$.*

Proof Assume $c \in S_{S3}^{D\cup O}$. Then $c \in \{c \mid DT \cup \{c\} \cup \vdash_3^{D\cup O} Obs\}$ which is a subset of $\{c \mid DT \cup \{c\} \cup \vdash_3^{\mathcal{C}\cup O} Obs\}$ by Theorem 9.2.0.6. Furthermore, $c \in D$ by Lemma 9.5.1.1. Therefore, the lemma is proven when we prove $c \in S_{W3}^{\mathcal{C}\cup O}$. This holds iff $DT \cup \{c\} \cup Obs \not\vdash_3^{\mathcal{C}\cup O} \bot$ iff $[DT]_3^{\mathcal{C}\cup O} \cup \{c\} \cup (Obs \cap O) \not\vdash \bot$ as an S-3-interpretation amounts to removing all clauses with letters outside the parameter S. In fact, this formula holds as $c \in S_{W3}^{D\cup O}$, i.e., $DT \cup \{c\} \cup Obs \not\vdash_3^{D\cup O} \bot$, which holds iff $[DT]_3^{D\cup O} \cup \{c\} \cup (Obs \cap O) \not\vdash \bot$. As clauses that do not contain c can be added to $[DT]_3^{D\cup O}$ without introducing an inconsistency, the result follows. □

We continue by proving the converse of the previous lemma that strong classification S_{S3}^{S} restricted to some part of the class hierarchy D (i.e., $S_{S3}^{\mathcal{C}\cup O} \cap D$) is a lower bound of the strong classification $S_{S3}^{D\cup O}$ (i.e., strong classification in which certain classes are removed from the parameter S.).

Lemma 9.5.2.2 *Let $D \subseteq \mathcal{C}$, and $O \subseteq \mathcal{O}$. Then $S_{S3}^{\mathcal{C}\cup O} \cap D \subseteq S_{S3}^{D\cup O}$.*

Proof Assume $c \notin S_{S3}^{D\cup O}$. Then either $c \notin S_{W3}^{D\cup O}$ or $DT \cup \{c\} \cup \nvdash_3^{D\cup O} Obs$. If $c \notin S_{W3}^{D\cup O}$, then $DT \cup \{c\} \cup Obs \vdash_3^{D\cup O} \perp$, which implies $DT \cup \{c\} \cup Obs \vdash_3^{\mathcal{C}\cup O} \perp$ (Theorem 9.2.0.6). Hence, $c \notin S_{W3}^{\mathcal{C}\cup O}$ and therefore $c \notin S_{S3}^{\mathcal{C}\cup O} \cap D$. Now assume the second case holds, i.e., $DT \cup \{c\} \cup \nvdash_3^{D\cup O} Obs$ and assume without loss $c \in D$. The formula holds iff $[DT]_3^{D\cup O\cup Obs} \cup \{c\} \cup \neg Obs \cup \nvdash \perp$. As clauses that do not contain c can be added to $[DT]_3^{D\cup O}$ without introducing an inconsistency it follows that $[DT]_3^{\mathcal{C}\cup O\cup Obs} \cup \{c\} \cup \neg Obs \cup \nvdash \perp$. However, the latter formula holds iff $DT \cup \{c\} \cup \nvdash_3^{\mathcal{C}\cup O} Obs$. Hence, $c \notin \{c \mid DT \cup \{c\} \cup \vdash_3^{\mathcal{C}\cup O} Obs\}$ and therefore $c \notin S_{S3}^{\mathcal{C}\cup O} \cap D$. □

Summarizing the inclusions proven in Lemmas 9.5.2.1 and 9.5.2.1 leads us to the following characterization of the effect of leaving out classes from the parameter S:

Theorem 9.5.2.3 *Let $D \subseteq \mathcal{C}$, and $O \subseteq \mathcal{O}$. Then $S_{S3}^{D\cup O} = S_{S3}^{\mathcal{C}\cup O} \cap D$.*

Proof This follows from Lemma 9.5.2.1 and 9.5.2.2. □

Theorem 9.5.2.3 states that the effect of leaving out classes from the parameter S results in a strong classification (using 3-entailment) restricted to those classes that are included in S. Furthermore, leaving out a class from S has no effect on the classes that are still included in S.

So far, we have only looked at the effect of leaving out classes from the parameter S. We continue this section by looking at the effect of leaving out observations from the parameter S. We will show that leaving out observations of S has no effect whatsoever on the solution set S_{S3}^S. Before doing so, we look at the effect of leaving out observations from S under the condition that S contains the set $\mathcal{C}$ of all classes. We prove that under this condition we obtain classical strong classification.

Lemma 9.5.2.4 *Let $\mathcal{C} \subseteq S$. Then $S_{S3}^S = S_S$.*

Proof $S_{S3}^S = \{c \mid DT \cup \{c\} \vdash_3^S Obs\} \cap S_{W3}^S = \{c \mid Obs \subseteq \{o \mid DT \cup \{c\} \vdash_3^S o\}\} \cap S_{W3}^S$ which by Lemma 9.4.2.1 is equal to $\{c \mid Obs \subseteq \{o \mid DT \cup \{c\} \vdash o\}\} \cap S_{W3}^S = \{c \mid DT \cup \{c\} \vdash Obs\} \cap S_{W3}^S$. As every element in $S_{W3}^S \setminus S_W$ is not an element of $\{c \mid DT \cup \{c\} \vdash Obs\}$ (Lemma 8.3.1.2) this is equal to $\{c \mid DT \cup \{c\} \vdash Obs\} \cap S_W$ which is the definition of S_S. □

Note that all ingredients are now proven to characterize S_{S3}^S. Suppose we have some parameter $S = D \cup O$ for some set $D \subseteq \mathcal{C}$ and $O \subseteq \mathcal{O}$. In Theorem 9.5.2.3 we proved that leaving out classes of the parameter S is the same as taking the intersection of the solution set $S_{S3}^{\mathcal{C}\cup O}$ with D. This solution set can be rewritten according to Lemma 9.5.2.4 to classical strong solution restricted to D. This characterization is stated in the following theorem.

Theorem 9.5.2.5 *Let $D \subseteq \mathcal{C}$ and $O \subseteq \mathcal{O}$. Then*

$$S_{S3}^{D \cup O} = S_S \cap D.$$

Proof By Lemma 9.5.2.4 and Theorem 9.5.2.3 we have $S_{S3}^{D \cup O} = S_{S3}^{\mathcal{C} \cup O} \cap D = S_S \cap D$. $\Box$

Note that this theorem states that the usefulness of approximating strong classification by using S-3-entailment is limited. There is no interpretation of S_{S3}^{S} possible in terms of strong classification in which missing or inconsistent attributes are allowed as there is a one to one correspondence between classes not in S and classes not in S_{S3}^{S}. Strong classification with S-3-entailment can only be used to restrict strong classification to a subset of the class hierarchy.

9.6 Summary

In Section 9.3, 9.4, and 9.5 we analysed the effect of the approximate entailment operators of Cadoli and Schaerf on the formalizations of classification we obtained in Chapter 8 with classes represented by necessary conditions. In this section we give an overview of the main results we obtained.

We started by analyzing the effect of both S-1- and S-3-entailment applied to weak classification. The use of S-1-entailment led to a complex classification method that we could not easily characterize. As strong- and explanative classification both used weak classification in their definitions, we therefore restricted the remainder of the analysis to the application of S-3-entailment to the various classification criteria,

For weak classification we proved first that it was unreasonable to leave out classes from the parameter S, because any class not in S would be part of the solution set S_{W3}^{S}. Second, under the assumption that $\mathcal{C} \subseteq S$, we proved that S_{W3}^{S} is exactly the same as weak classification that allows for certain attributes to have an inconsistent value. This follows from the formula we obtained in Theorem 9.3.2.6:

$$S_{W3}^{\mathcal{C} \cup (\mathcal{O} \setminus O_{incorrect})} = S_{W}^{incorrect}.$$

For explanative classification we proved that the approximation obtained by using S-3-entailment is not useful as the approximation obtained led to an uninformative sequence of approximating sets of the solution set S_{E3}^{S}. This follows from the following formula we obtained in Theorem 9.4.2.2

$$S_{E3}^{(\mathcal{C} \setminus D) \cup O} = S_E \cup D,$$

for some set $D \subseteq \mathcal{C}$ and some set $O \subseteq \mathcal{O}$.

For strong classification we proved that the approximation obtained by using S-3-entailment is exactly the same as classical strong classification restricted to a part of the class hierarchy. This follows from the following formula we obtained in Theorem 9.5.2.5

$$S_{S3}^{D \cup O} = S_S \cap D,$$

for some set $D \subseteq \mathcal{C}$ and some set $O \subseteq \mathcal{O}$.

9.7 Conclusions

This study began with the premise that methods are needed that are robust enough to deal with incorrect and incomple data. In particular many AI tasks would benefit from such a robustness property as the knowledge and data used by such tasks are often inherently incorrect and incomplete for some part. Furthermore, general approximation methods are preferred as new research areas will create new tasks which could benefit by being more robust in dealing with incorrect and incomplete information.

A general technique for approximating logical inference problems one can use is replacing the standard entailment operator by the approximate entailment operators developed in [Schaerf and Cadoli, 1995]. Although this technique is general and has some desirable properties, little is known about the method when applied to specific problems. This chapter analysed the applicability of the method of Cadoli and Schaerf to classification.

The main results of this chapter are the formulas obtained that describe the effect of replacing the entailment operator by the S-3-entailment operator for three of the classification forms. It was proven that, using S-3-entailment, approximate weak classification behaves identical to weak classification that allows for certain inconsistencies. Furthermore, approximate strong classification behaves identical to strong classification restricted to a subset of all classes. The solutions to approximate explanative classification are identical to classical explanative classificaton to which a set of classes is added.

Although approximations constructed using S-1- and S-3-entailment satisfy a number of properties desirable for any approximation method (Section 1.3.1), the usefulness of the method for practical applications remains questionable.

First, the theoretical analysis done so far, shows that the usefulness of the approximate entailment operators is limited when applied to the task of classification. Only S-3-entailment applied to weak classification leads to an approximation that can be interpreted in terms of missing or inconsistent attributes. S-3-entailment applied to strong- and explanative classification leads to obvious and uninformative approximations that can be written as a union or intersection of sets.

Second, obtaining a clear theoretical analysis of the effect of applying S-1- or S-3-entailment to some task may be hard if not impossible. Even for a task as classification with an easy and limited structure we were not able to obtain a clear description of the effect of S-1-entailment applied to the three classification forms.

Third, the method of S-1- and S-3-entailment uses a parameter S containing propositional letters. The accuracy of the approximation can be increased by adding more letters to S. The method of Cadoli and Schaerf does not provide any information about how to choose the letters that should be added to S and the order in which this should be done.

The parameter S should be chosen in such a way that the obtained approximation behaves like an anytime algorithm (Section 1.3.1).

The last point shows that a theoretical anlysis is not always sufficient to obtain all the information needed for a method (like S-1- and S-3-entailment) to be applicable for practical problem solving . The following chapter looks at empirical analyses for obtaining such information for S-1- and S-3-entailment.

Chapter 10

Empirical analysis

10.1 Introduction

Chapter 9 analyzed the approximations by using the definitions of the method of Cadoli and Schaerf [Schaerf and Cadoli, 1995] together with some of its properties, and the rules of propositional logic. The benefit of this approach is that a formula obtained using this kind of analysis can give a very precise description of the effect of replacing the entailment operator by either S-1- or S-3-entailment. However, this approach also has some drawbacks.

First, it may not be possible to find a formula describing the effect of replacing the entailment operator by either S-1- or S-3-entailment. Such a formula may not exist, or it may be too difficult to obtain. For example, the analysis of Chapter 9 does not provide any formula for any approximation that contains S^S_{W1}, i.e., approximate weak classification using S-1-entailment.

Second, even if a description in the form of a formula is obtained, it may not contain all necessary information. For the method of Cadoli and Schaerf to be applicable one also needs to know how to choose the parameter S, i.e., how to determine which letters should be added to S and in which order. Furthermore, information is needed about the quality of the approximation and how fast the approximation improves over time.

The analysis performed in Chapter 9 should therefore be done together with an empirical analysis of the various approximations. Such an empirical analysis may result in new insights for approximations that could not be clearly described using an theoretical analysis. Furthermore, an empirical analysis may provide guidelines for choosing the parameter S and give quantitative information about the approximation methods (e.g., the growth rate of the set of solutions when S increases).

The rest of this chapter is structured as follows. Section 10.2 discusses some questions that should be answered before any empirical analysis can be started. Thereafter, results of two empirical analyses are presented. Section 10.3 gives results of approximate strong classification using S-3-entailment on a constructed theory with various orderings of S. The goal of this section is to provide empirical results that corroborate the results of Chapter 9. Section 10.4 also gives results about approximate strong classification using S-

3-entailment, but validates an ordering of S that is based on certain properties of a theory. The chapter ends by giving conclusions in Section 10.5 and future work in Section 10.6.

10.2 Preliminary questions

To quantitatively analyze the approximations obtained by using S-1- or S-3-entailment, a number of questions should be addressed before any experiment can be started.

The goal. For what reason do we perform the quantitative analysis? Several reasons can be given for performing a quantitative analysis. For example, to corroborate a formula obtained by analysing the logical formulas, collect data from which a formula may be constructed (as a hypothesis), or to obtain quantitative information not present in any formula obtained yet.

The domain. Which parameters of a domain may influence the results of an empirical analysis? As different domain theories may result in different quantitative properties for the approximations one needs to identify those parameters of a theory that may influence the outcome of an empirical analysis. For example, when a domain theory contains clauses of the form $c \rightarrow a_1 \vee \ldots \vee a_n$ then n is a parameter as this has effect on the length of a clause.

The parameter S. Which letters should be added to the parameter S and in which order? To run experiments with a parameter S, which increases by adding new letters, one needs to specify in which order letters are added to S and how many letters are added to S at each iteration.

Presentation. How do we present the results of the empirical analysis? The method that has been approximated using S-1- or S-3-entailment returns an approximate result. The question rises how to present for some or each parameter choice of S the corresponding approximate results. One should keep in mind that the empirical analysis may result in a large set of data. This data should be presented in such a way that it is clear if the goal of the analysis is obtained.

10.3 Experiment 1

Consider the case of strong classification as this form of approximate classification (together with explanative classification) imposes the most constraints on a domain theory DT. (In particular, this section looks at approximate classification constructed by using S-3-entailment.) A theory DT that can be used for strong classification consists of the following two types of clauses:

$$c \rightarrow a_i,$$

$$a_1 \rightarrow \neg a_2 \wedge \ldots \wedge \neg a_n.$$

The first clause is part of a class description. The second clause ensures that each attribute can only have one value at a time. Note that $DT \cup \{c\}$ is assumed to be consistent, which means that DT cannot include $c \rightarrow a_i$ and $c \rightarrow a_j$ for $i \neq j$ (i.e., a class c cannot have two different values for the same attribute). Before continuing with an empirical analysis we will consider the questions posed in the previous paragraph.

In the case of strong classification approximated by using the S-3-entailment operator, a formula can be given that clearly describes the effect of replacing the entailment operator by S-3-entailment (Theorem 9.5.2.5). It states that adding observations to S will leave the approximation unchanged. Adding a class to S either adds the class to the set of approximate solutions or leaves the set of approximate solutions unchanged (depending on whether the class is a solution for classical strong classification). Based on these observations, the goal of the empirical analysis of this section is to collect data that corroborates Theorem 9.5.2.5.

Considering the question about the structure of the domain one can identify parameters such as:

1. The number of values for an attribute as this directly influences the length of the clauses for ensuring the exclusion of multiple values for an attribute.
2. The number of attributes for a class (e.g., $c \rightarrow a_i$ and $c \rightarrow b_j$ with $a \neq b$) as this influences the number of clauses for each class.
3. The number of classes for an attribute (e.g., $c \rightarrow a_i$ and $d \rightarrow a_j$ with $c \neq d$) as any operation on an attribute (e.g., removal) will influence the classes occurring in a clause with that attribute.

However, the empirical analysis of this section is used to corroborate Theorem 9.5.2.5. No quantitative information like for example the growth rate of S_{S3}^{S} for increasing S is obtained. These parameters are therefore not important for the empirical analysis of this section.

Considering Theorem 9.5.2.5 and the question about the parameter S, it follows that any ordering of the classes can be used. Various orderings on the classes should give the same results. Adding a class c to S results either in S_{S3}^{S} remaining the same or c being added to S_{S3}^{S}. Theorem 9.5.2.5 also states that adding observations to S does not change the solution set S_{S3}^{S}. This can be tested by taking an ordering of the classes and experiment by placing the observations either at the end or at the beginning.

For the presentation a table can be used for presenting the results. As these experiments are used to corroborate Theorem 9.5.2.5, it should be clear from this table and its context which classes are added to S, which classes are returned by the approximation, and which of those classes belong to the solution set S_S of classical strong classification. The size of a table can be reduced by only presenting the parameters that results in an actual change of a solution set.

10.3.1 The experiment

To run the experiment a theory (i.e., a domain of classes and attributes) is needed. As noted in the previous section, for this experiment it does not matter which parameters are used for the construction of a theory as these cannot influence the corroboration of a qualitative result. However, the theory should at least ensure a reasonable number of classical strong solutions in a domain not too large. For some parameters a reasonable choice should therefore be made beforehand while the remaining parameters can be chosen at random.

In Chapter 8 it was observed that for a class c to be a strong solution for some set Obs of observations it must hold that $A_{Obs} \subseteq A_c$ (just below Definition 8.2.3.5). It follows that a class is more likely to be a strong solution when A_{Obs} is small and A_c is large. Hence, a class description should have many AV-pairs and the object description should have few AV-pairs. Furthermore, for a class to be a strong solution there must be a match between values of similar attributes in the object description and in the class description. This match is more likely if attributes are not allowed too many values.

In our experiment the parameters 'number of classes and attributes', 'minimum and maximum attributes for each class', 'and minimum and maximum number of values for each attribute' were chosen beforehand while the rest of the theory was created at random. This resulted in a theory with 20 classes, 5 attributes (one attribute with 2 values and the rest with one value), and 1 observation for the object description.

The implemented algorithm of approximate strong classification uses an external call to a SAT solver which accepts files in the DIMACS cnf format.[i] Therefore, all classes and AV-pairs are represented by numbers in the implementation. Numbers 1 through 20 represent classes and numbers 21 through 26 represent AV-pairs with 21 the AV-pair used in the object description. For a clearer reading we will write in the remainder of this chapter c_i for some i that represents a class, and o_i for some i that represents an AV-pair. In the following the shorthand 3–7 is used to denote the sequence of numbers 3,4,5,6, and 7. Running a classic weak- and strong classification algorithm on the theory plus a single observation results in the following solution sets:

$$\begin{aligned} S_W &= c_{1-3}, c_{5-10}, c_{13-17}, c_{19}, \\ S_S &= c_1, c_3, c_5, c_{8-10}, c_{13-16}, c_{19}. \end{aligned}$$

Note that $S_S \subseteq S_W$ holds.

Now all ingredients, except for the parameter S and an ordering on S, are present to investigate the approximate version of strong classification with S-3-entailment with an empirical analysis. As weak classification is part of the definition of strong classification, attention is first given to this approximation before looking at approximate strong classification.

[i] ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc

10.3.1.1 Approximate weak classification

The approximate weak classification algorithm S^S_{W3} is first applied to the sequence c_{1-20},o_{21-26}, i.e., the algorithm starts with the parameter $S = \{c_1\}$ and adds the elements in the order of the sequence c_{1-20},o_{21-26} to S one at a time at the start of each iteration of the algorithm. Hence, first the classes are added in the order c_{1-20}, then the object description o_{21} is added, and finally the other AV-pairs o_{22-26} are added. The results are shown in Figure 10.1. Note that at the end of the ordering, when S includes all letters of the alphabet, S^S_{W3} should be equal to classical weak classification S_W. In figures that present results of the empirical analysis this is always shown in the bottom line of the figure.

S	S^S_{W3}
c_1	c_{1-20}
c_{1-20},o_{21}	c_{1-3}, c_{5-10}, c_{13-17}, c_{19}

Figure 10.1: S^S_{W3} with order c_{1-20},o_{21-26}.

Note that only those choices for the parameter S are shown that change the approximation S^S_{W3} when compared to the previous iteration. Hence, Figure 10.1 should be read as follows. The first iteration of S^S_{W3} is with $S = \{c_1\}$. As nothing has been computed before, this line is shown in Figure 10.1 and it states that the algorithm returns all 20 classes. Then for a number of iterations of the algorithm nothing happens until o_{21} is added to S. As Figure 10.1 shows, the output of S^S_{W3} changes from 'c_{1-20}' to 'c_{1-3}, c_{5-10}, c_{13-17}, c_{19}', which is equal to S_W.

A number of observations can be made about Figure 10.1, which also hold when tested with other orderings for S. First, adding the AV-pairs o_{22-26} does not result in any changes. Theorem 9.3.2.6 states that S^S_{W3} is the same as weak classification that allows inconsistencies for those AV-pairs that are not in S. As there is no consistency check for AV-pairs that do not belong to the object description, it does not matter if these AV-pairs are included or excluded from S.

Second, Figure 10.1 indicates that the AV-pair o_{21} may be important as adding this value results in a change of the solution set S^S_{W3}. This observation is in fact already stated in Lemma 9.3.1.5. When $S \cap Obs = \varnothing$ (i.e., in this case $o_{21} \notin S$) then S^S_{W3} includes all classes. As not much can be observed from one run of the approximate weak classification algorithm, it should also be run using other orderings. For example, running the algorithm on the ordering c_{1-20},o_{22-26},o_{21} shows behaviour similar to Figure 10.1. When o_{26} is added to S, S^S_{W3} still contains all classes, but when o_{21} is added, S^S_{W3} becomes equal to S_W.

Putting o_{21} in front of the ordering, (e.g., as in the ordering o_{21-26},c_{1-20}), results in S^S_{W3} undergoing more changes (Figure 10.2). Note that these results illustrate Lemma 9.3.1.4. Any class not in S is included in S^S_{W3}, but as soon as this class is added to S and found to be inconsistent it is removed from S^S_{W3}. For example class c_{11} is added to S in

S	S^S_{W3}
o_{21}	c_{1-20}
o_{21-26},c_{1-4}	c_{1-3}, c_{5-20}
o_{21-26},c_{1-11}	c_{1-3}, c_{5-10}, c_{12-20}
o_{21-26},c_{1-12}	c_{1-3}, c_{5-10}, c_{13-20}
o_{21-26},c_{1-18}	c_{1-3}, c_{5-10}, c_{13-17}, c_{19-20}
o_{21-26},c_{1-20}	c_{1-3}, c_{5-10}, c_{13-17}, c_{19}

Figure 10.2: S^S_{W3} with order o_{21-26},c_{1-20}.

line three and is also removed from S^S_{W3} in line three. Note that the observations o_{21-26} are first added, and therefore no inconsistencies are allowed for any class (Theorem 9.3.2.6).

Note that from an anytime perspective Figure 10.2 should be preferred over Figure 10.1. The change of the solution set S^S_{W3} is more gradual in Figure 10.2 than in Figure 10.1. Hence, from the previous observations one can conclude that an ordering that puts the AV-pairs of the object description in front is preferred (from an anytime perspective) when used in the computation of S^S_{W3}.

Sofar, only S^S_{W3} has been discussed. The rest of this section looks at S^S_{S3}.

10.3.1.2 Approximate strong classification

The same empirical analysis, i.e., using the same theory and same choices for the parameter S, were also performed for S^S_{S3}. The results of this analysis are shown in Figure 10.3.

S	S^S_{S3}
c_1	c_1
c_{1-3}	c_1, c_3
c_{1-5}	c_1, c_3, c_5
c_{1-8}	c_1, c_3, c_5, c_8
c_{1-9}	c_1, c_3, c_5, c_{8-9}
c_{1-10}	c_1, c_3, c_5, c_{8-10}
c_{1-13}	c_1, c_3, c_5, c_{8-10}, c_{13}
c_{1-14}	c_1, c_3, c_5, c_{8-10}, c_{13-14}
c_{1-15}	c_1, c_3, c_5, c_{8-10}, c_{13-15}
c_{1-16}	c_1, c_3, c_5, c_{8-10}, c_{13-16}
c_{1-19}	c_1, c_3, c_5, c_{8-10}, c_{13-16}, c_{19}

Figure 10.3: S^S_{S3} with order c_{1-20},o_{21-26}.

Figure 10.3 shows the same results as stated by Theorem 9.5.2.5. A class is only given as output when this class is included in the parameter S and is a solution according to classical strong classification. Using other orderings for the algorithm on the same theory (data not shown here) results in similar behaviour.

Further note that in Figure 10.3 adding AV-pairs to S has no effect on the solution set S^S_{S3}. Using other orderings for the algorithm on the same theory (data not shown here) results in similar behaviour.

10.4 Experiment 2

The method of Cadoli and Schaerf uses a paramter S resulting in a whole spectrum of approximations that range from zero to optimal precision. The practical usefulness of the method therefore depends on the choice for S, making this choice a crucial part of the method. Currently, the method has not been evaluated beyond diagnosis in [tenTeije and vanHarmelen, 1996, tenTeije and vanHarmelen, 1997] and belief revision [Chopra *et al.*, 2001] by means of a quantitative and qualitative analysis. In this section heuristics are presented for the parameter S for strong and explanative classification and validated against random orders of S.

Section 10.4.1 presents the experimental analysis of strong classification. Section 10.4.2 presents the experimental analysis of explanative classification. Note that in Chapter 9 a number of restrictions for reasonable choises for S were obtained. Hence, the theoretical analysis of Chapter 9 can be used to set boundaries for choices for S. One such boundary is leaving out observations from S. In Chapter 9 it was shown that adding observations to S does not influence the approximations for strong classification and explanative classification. Hence, in the following experiments we can restrict the order of S to an order of the classes.

10.4.1 Approximate strong classification

With approximate strong classification the set of solutions is approximated from below (sound but incomplete), i.e., by adding more classes to S more strong solutions are obtained. A reasonable choice for S therefore seems to be to prefer a class c over a class d when class c is more likely to be a strong solution. As for strong classification $A_{Obs} \subseteq A_c$ must hold (i.e., all attributes in Obs must also occur in c and their values must match), this seems to be the case when (1) the number of attributes in the class description of c is higher than the number of attributes in the class description of d, and/or (2) the number of possible values that can be assigned to attributes of c is less than the number of possible values that can be assigned to attributes of d. (More precisely, the second heuristic is computed by taking the product of the number of possible values per attribute.)

These two heuristics lead to four possible orders. S_1: apply only the first heuristic, S_2: apply only the second heuristic, S_3: apply the first followed by the second heuristic (in case two classes have the same number of attributes), and S_4: apply the second heuristic followed by the first heuristic (in case two classes have the same product of the possible values per attribute).

10.4.1.1 The experiment

For the experiment of strong classification a larger theory was created than the one used in the first empirical analysis. To obtain a reasonable number of classical strong solutions some conditions were set for the theory while the rest of the theory was created randomly. The theory consisted of 100 classes and 10 attributes. The maximum allowed number of values for an attribute was set at 5 and the class descriptions contained between 5 and 10

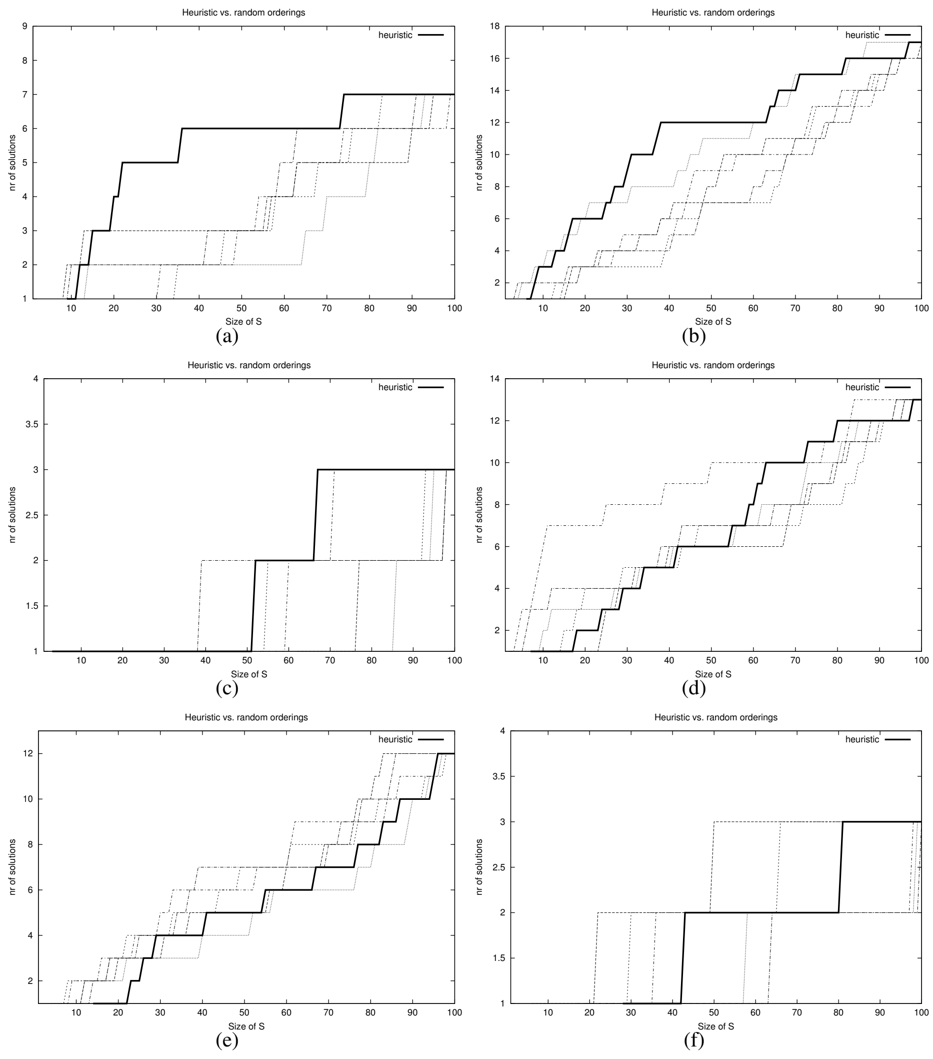


Figure 10.4: Results of heuristic order S_3 against 5 random orders.

attributes. Furthermore, 30 random object descriptions were created containing one, two, or three observations.

The goal of the experiment is to validate the heuristic orders $S_1, \ldots, S_4$ against each other and against random orders. An order o_1 is preferred over an order o_2 if the approximating behaviour of the approximation S_{S3}^{S} has a better anytime performance profile using order o_1 then when using order o_2 for the parameter S.

Some results of the order S_3 against 5 random orders are shown in Figure 10.4. The results of the experiment are presented as follows. Each subfigure shows the results of the orders on one specific object description. On the x-axis the size of the parameter S is given (i.e., the number of classes) and on the y-axis the number of solutions is given (i.e., the number of classes included in the solution set S_{S3}^{S}. Note that for different object descriptions the total number of classical strong solutions is usually different. The scale of the y-axis will depend on the maximal number of classical strong solutions and will therefore be different for the various subfigures.

Comparing the heuristic order S_3 against the random orders, one can identify three different kind of comparisons. The heuristic order performs better on Figures 10.4(a), 10.4(b), and 10.4(c). The heuristic order performs nor better nor worse on Figure 10.4(d). In the first part of the graph, the random orders perform better, but in the last part of the graph the heuristic order performs better. The heuristic order performs worse on Figures 10.4(e) and 10.4(f).

Note that from an anytime perspective Figure 10.4(a) shows the preferred behaviour of an anytime algorithm. The solution set grows monotonically, which ofcourse also holds for the other figures, but the graph also shows a diminishing return (i.e., most of the solutions are generated at the beginning).

Sofar only the approximating behaviour is obtained for specific object descriptions. More interesting is to compute the *average* approximating behaviour of the various orders on a set of observations. As the number of strong solutions will be different for different object descriptions, each result on one specific object description needs to be normalized before the average can be computed. In the following the quality of the approximate strong classification algorithm will be mapped

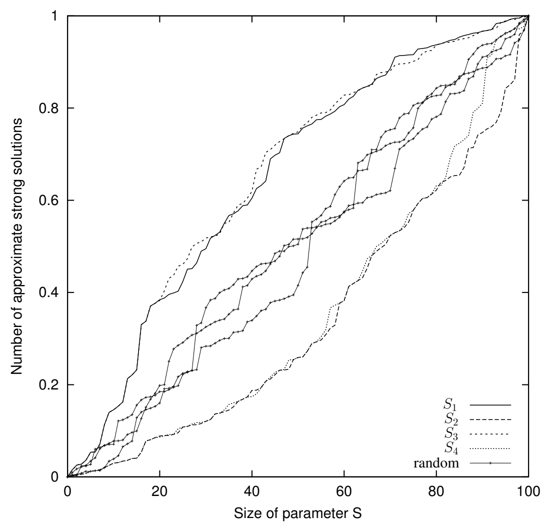


Figure 10.5: Results of orders $S_1, \ldots, S_4$ and three random orders in approximate strong classification.

on the interval [0,1] by dividing the obtained value through the maximum value that can be obtained, i.e., the number of classical solutions.

Figure 10.5 shows the results of the orders $S_1, \dots, S_4$ as well as three random orders. More random orders were tested in practise, which resulted in similar behaviour as shown in Figure 10.5, but they are left out for readability.

The results show that the order that selects classes with the highest number of attributes, i.e., S_1, performs much better than a random order for S. Figure 10.5 also shows that S_1 can be further improved by also considering the number of possible values, i.e., S_3. The orders that first select the classes with the lowest number of possible values, i.e., S_2 and S_4 perform much worse than a random order of S. This may seem surprising at first, but not when considering the results of order S_1. Classes with the lowest number of possible values are often classes with less attributes (as one less attribute means one less number in the product used to compute the second heuristic), hence orders S_2 and S_4 tend to take the opposite order of S than S_1.

10.4.2 Approximate explanative classification

With approximate explanative classification the set of solutions is approximated from above (complete but unsound), i.e., by adding more classes to S more incorrect solutions are discarded. A reasonable choice for S therefore seems to be to prefer a class c over a class d when class c is less likely to be an explanative solution. As for explanative classification $A_c \subseteq A_{Obs}$ must hold, this seems to be the case when (1) the number of attributes in the class description of c is higher than the number of attributes in the class description of d, and/or (2) the number of possible values that can be assigned to attributes of c is higher than the number of possible values that can be assigned to attributes of d.

These two heuristics lead to four possible orders. E_1: apply only the first heuristic, E_2: apply only the second heuristic, E_3: apply the first followed by the second heuristic (in case two classes have the same number of attributes), and E_4: apply the second heuristic followed by the first heuristic.

The goal of the experiment is to compute the average approximating behaviour of the various orders. As the set of classic solutions is approximated from above (incorrect solutions are discarded when S increases) normalization of the result of each object description is more complicated than the case of strong classification. To obtain an increasing quality function on the interval $[0, 1]$ as in Figure 10.5 we apply for a theory with n classes the normalization $(n - v(i, o, s))/(n - v(n, o, s))$ where $v(i, o, s)$ is the size of the solution set at iteration i for some object description o and some chosen order s. Note that $v(n, o, s)$ is equal to the number of classical explanative solutions.

The theory used for the experiment consisted of 100 classes and 10 attributes. The maximum number of allowed values per attribute was set at 7 and the class descriptions contained between 1 and 3 attributes. Furthermore, 30 random object descriptions were created consisting of eight, nine, and ten attributes. The results of the experiment are shown in Figure 10.6.

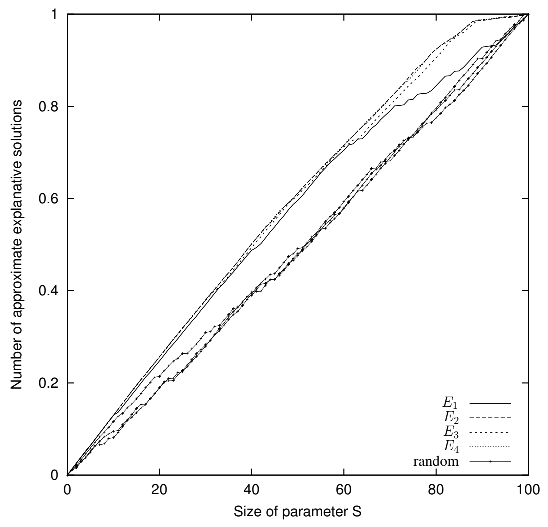


Figure 10.6: Results of orders $S_1, \ldots, S_4$ and three random orders in approximate explanative classification.

In this experiment, all heuristic orders produce a better approximating behaviour when compared with the random orders. However there is little to choose between the various heuristic orders. Order E_1, which only considers the number of attributes, performs worst, but it can be improved by also considering the number of values (E_3). The best orders are E_2 and E_4. Note that classes with many possible values for its attributes are often also classes with many attributes. Hence, the second heuristic tends to prefer classes also preferred by the first heuristic. Therefore, all orders based on the two heuristics result in very similar approximating behaviour.

Note that similar experiments on a different theory resulted in similar behaviour as in Figure 10.6. However, the difference between the various orders $E_1, \ldots, E_4$ was even less, probably caused by a lower number of explanative solutions.

10.5 Conclusion

Using a theoretical analysis does not allways provide all the necessary information about a method. In particular, the analysis of Chapter 9 does not provide guidelines for choosing which letters should be added to the paramter S and in which order. This chapter fills in some of the gaps of Chapter 9 by using empirical analyses of the approximate classification forms constructed by using S-1- and S-3-entailment.

This chapter first discussed the kind of questions one needs to answer before any experiments can be started. Thereafter it presented two empirical analyses, one for providing corroboratory results for the theoretical results of Chapter 9 and one for validating a particular choice of S for a given theory.

Although an empirical analysis is useful to accompany a theoretical analysis, some drawbacks of this approach were found. Although some of these may be obvious, one should be aware of these drawbacks.

First, results of an empirical analysis may depend on the particular instance used for analysing a method. For example, in this chapter, empirical results may depend on the theory used for classification. Determining the structure of such a theory and representing it as a set of parameters may be hard if not impossible.

Note that the experiments performed in this chapter did not depend on any parameter of the domain. This problem therefore did not need to be addressed. In Section 10.3 we mentioned some parameters that may be important when performing an empirical analysis of approximate classification. However, there is no indication if the given list of parameters is complete and fully characterizes the domain or identifies all possible influences on experimental results.

Second, even if a domain can be characterized as a set of parameters, there may be too many parameters to explore using an empirical analysis.

Third, if not all instances of a problem can be used in an empirical analysis, then the results of this analysis cannot be used as facts. An empirical analysis can be used to corroborate a statement or theory and provide insight in a problem. However, an empirical analysis cannot be used to prove a property of a method unless all instances are explored.

10.6 Future work

A theoretical analysis as well as an empirical analysis have been presented for the approximate entailment operators of [Schaerf and Cadoli, 1995]. Both of these analyses allow room for further research.

The theoretical analysis of Chapter 9 gives a detailed analysis of S-3-entailment used in classification. However, the analysis can be expanded in two ways. First, the analysis of Chapter 9 is restricted to classes defined by necessary conditions. The other representations mentioned in Section 8.3 should be included in future research. Second, all obtained results are for S-3-entailment. No results are obtained for S-1-entailment. This form of approximation led to a complex formula when used in weak classification that could not easily be described. Nevertheless, it should be considered for future work. Another approach might be to accompany the theoretical analysis of S-1-entailment used in classification with an empirical analysis.

The empirical analysis of this chapter presented two analyses, which can be expanded in a number of ways. The second analysis (Section 10.4) validated a particular choice of ordering on S against random orderings on a given theory. One could try to construct other kind of orderings and validate these against the proposed ordering. Note that the theory used in the experiment was random. Other orderings might be preferable when more is known about a theory. For example, when an object description may only contain positive observations.

Another interesting path to explore is the use of another logic. This part of the dissertation focussed mainly on formalizations of classification in propositional logic. However, the method can also be used for formalizations in first-order logic.

Last but not least, other tasks should be considered, which can be formulated in logic, that may be useful when approximated by using the approximate entailment operators. The more so, as the usefulness of the approximate entailment operators applied to classification is limited. The analyses presented in this chapter, i.e., the combination of theoretical and empirical analyses, can be used as a guide to perform similar studies of the approximate entailment operators applied to other domains.

Part IV

Contributions and Discussion

Chapter 11

Contributions and discussion

11.1 Contributions

This dissertation addresses the following research goal:

> "A theoretical and empirical analysis of the effect of approximation on symbolic problem solving."

The approach to this research goal was to analyse three specific areas in which approximation is present or could be used in a promising way in symbolic problem solving. These areas were: I. Knowledge-Based Systems, II. Knowledge Compilation, and III. Approximate Entailment. More specifically, in these three areas respectively, this dissertation looked at the following research goals:

1. Quantifying the robust behaviour of KBSs in the presence of incorrect and incomplete knowledge or data.
2. Analyse the applicability of knowledge compilation techniques to planning problems.
3. Analyse the applicability of the approximate entailment operator to classification problems.

The contributions of this dissertation that followed from these research goals were already mentioned in Section 1.4. This section describes those contributions in more detail.

Part I: Methodology for measuring the robustness of KBSs. (Chapter 2) After arguing the need for quantitative analysis of KBSs, Chapter 2 describes a methodology for measuring the robustness of KBSs with respect to some input quality (e.g., correctness of knowledge or data input). The methodology is based on the notion of degradation studies: analyse how the quality of the output changes as a function of degrading input (e.g., knowledge base, data input). For measuring the output quality, the methodology uses the measures recall and precision, which are well known measures from the field Information Retrieval. The generality of these measures makes them applicable to a wide range of KBSs. The measurement of the input quality is deliberately left open because it depends on the system under study. However, two aspects (correctness and completeness) are proposed for consideration. The interpretation of the results of the robustness analysis are also left open because the interpretation depends on the goal of the robustness analysis and the system under study. Chapter 2 proposes a number of competing definitions for comparing the robustness results of different systems (or different variations of a system) on the same degrading.

Part I: Application of the proposed methodology to a particular KBS. (Chapter 3) Besides claiming to have a methodology for measuring the robustness of KBSs with respect some input quality, a case study is performed in which the methodology is applied to a realistic KBS for classifying commonly occurring vegetation in Southern Germany. In this case study the robustness is measured with respect to incomplete data input as well as an incomplete and incorrect knowledge base. This case study shows that the methodology is attainable when applied in practice and that the methodology can lead to a better insight of the system under study.

Part II: Empirical analysis of the use of knowledge compilation for planning problems. (Chapter 7) As it is still unclear what can and cannot be achieved through knowledge compilation, empirical analyses are needed to show the applicability of knowledge compilation to practical problems. Chapter 7 reports the results of experimentation with knowledge compilation. First, the three main parameters in this kind of empirical research are identified. The main parameters are the problem domain, the encoding of the problem domain, and the knowledge compilation method used. The specific choices that are made for the empirical analysis of Chapter 7 are to look at the domain of planning problems, which are translated from PDDL into propositional logic using the Medic planner. The knowledge compilation methods that are used are Zres and Directional Resolution. Various parameters are investigated in the empirical analysis (e.g., encoding, adding domain specific knowledge, different planning problems) and although there is some difference in results for various parameter choices, no substantial gain is obtained through knowledge compilation. More precisely, all the results that are obtained in the empirical analysis are negative. At the very least this shows that making knowledge compilation practical is non-trivial.

Part III: A formal analysis of classification, including forms of approximate classification. (Chapter 8) Using the systematic set-theoretic description of classification given in [Jansen, 2003] we formalize classification in propositional logic. By varying the comparisons between attributes in the object description and attributes in the class descriptions various forms of classification are obtained. These classification forms are formalized in propositional logic under various representations for the class descriptions. The chapter concludes with two intuitive forms of approximate classifications, namely classification that allows missing or incorrect attributes. These approximations are also considered in [Jansen, 2003], but in this dissertation we give a formalization in propositional logic and make a clear distinction between missing or incorrect attributes occurring in either the object description, class description, or both.

Part III: A formal analysis of the use of approximate deduction for classification reasoning. (Chapter 9) Classification is approximated by using the approximate entailment operator of Cadoli and Schaerf in the logical definitions given in Chapter 8. These approximations are analysed using propositional reasoning and properties of the approximate entailment operator. This results in three formulas, which clearly describe the effect of the approximate entailment operator. In particular the approximation of weak classification using S-3-entailment is shown to be equivalent to weak classification that allows for inconsistent attributes. The approximation of strong classification using S-3-entailment is shown to be equivalent to strong classification restricted to a part of the class hierarchy. The approximation of explanative classification using S-3-entailment is shown to be equivalent to explanative classification with a union of some classes.

Part III: Empirical analysis of approximate classification through approximate deduction. (Chapter 10) This chapter argues the need for an empirical analysis of the method of Cadoli and Schaerf. Certain information like which letters and in which order those letters should be added to the parameter S cannot be obtained through a theoretical analysis. After discussing the kind of questions one need to answer for an empirical analysis of the method of Cadoli and Schaerf two empirical analyses are presented. The first analysis is done for obtaining information that corroborates the theoretical results obtained in Chapter 9. The second analysis is done for validating a particular ordering of the parameter S on a given theory, i.e., the kind of information that could not be obtained through the theoretical analysis used in Chapter 9.

11.2 Discussion

Although there has already been done much research on the topic of approximation in general, there are still many question left to be answered. This also holds when we restrict our attention to symbolic problem solving. The lack of a numeric measure in a logical problem that tells us how much an approximate solution differs from the correct solution means that all research on numeric problem approximation cannot directly be applied when tackling symbolic problem solving through approximation.

The quality of an approximation of a logical problem is usually stated in terms of soundness and completeness. Such a quality measure is qualitative. Considering some algorithm for an inference problem, it may state for example that all solutions given as output are correct, but may be some correct solutions are not given as output. Quantitative information can often not be given using such measures, e.g., how many correct solutions are not given as output.

Therefore, theoretical analysis should be accompanied by empirical analysis. There is often a lack of attention to this kind of research. New methods and techniques are developed one after another and are shown to outperform its predecessors either on some small set of examples or by means of a worst-case complexity analysis. Although such analysis is useful it does not make those techniques applicable for practical problem solving.

One can identify a number of problems in current research that hinder the applicability of techniques in a practical setting. First, methods are often tested on problems that are either too small or too artificial. Hence, it will be unclear if the method can be applied to larger problems (i.e., is the method scalable) or if the method can be applied to other, realistic domains.

Second, empirical analysis is done on a limited scale and is sometimes even lacking for certain methods and techniques. In the latter case the reason for the lack of empirical analysis may be because either no one has ever bothered to perform an empirical analysis, or someone *did* perform an empirical analysis, but did not (or could not) communicate the results to other researchers. Note that the lack of communication especially holds when the results of an empirical analysis are negative. This believe corresponds with [McDermott, 1981], as one of the statements made in this article is:

> "AI as a field is starving for a few carefully documented failures."

A reason why it is difficult to make a 'failure' sound scientifically interesting is the need for generality. Very often the result is "our system failed to carry out task X". But what can one conclude from this, other than that this particular implementation is in some way not up to the task X? In conventional physical science, there is a well-defined notion of an experiment being based on (exemplifying, testing, etc.) a theory. And theories are (should be) clearly stated in terms which are both well-defined and (ultimately) grounded in empirical data. What theories are there in AI or in a subfield like symbolic problem solving? What does it mean for a program to embody a theory, and not to embody other complicating factors (i.e., to be a controlled experiment)?

Third, even if an extensive formal analysis has been done it may not be the right kind of analysis for practical purposes. For example, a worst-case analysis does not give information about the average behaviour of an algorithm. Some algorithms may be useful in practise even if their worst-case behaviour is unsatisfactory, because they exhibit good behaviour on average and the worst-case scenario is rarely encountered.

Note that we do not claim that these problems are easy to overcome. The nature of empirical analysis is that there are often loose ends. This holds especially when obtained results are negative. Were these negative results because of the wrong method used, the wrong representation used, or some other parameter? Often there are a lot of parameters making it difficult if not impossible to explore all options. This also makes it hard to publish and communicate results between researchers. Clearly, empirical analysis would benefit from guidelines and sets of benchmarks for particular techniques and problem domains.

Notation

$\mathcal{C}$	The set containing all classes
D	A set containing classes, i.e., $D \subseteq \mathcal{C}$
c, d	A class, i.e., $c, d \in \mathcal{C}$
$\mathcal{A}$	The set containing all attributes
A	A set containing attributes, i.e., $A \subseteq \mathcal{A}$
a, b	An attribute, i.e., $a, b \in \mathcal{A}$
$\langle a, v\rangle$	An attribute-value pair (AV-pair), also referred to as an observation
a_1	Shorthand for the AV-pair $\langle a, 1\rangle$
a_u	Shorthand for the AV-pair $\langle a, unknown\rangle$
A_c	Given some class c, $A_c = \{a \mid \langle a, v\rangle \in c\}$, i.e., the set of all attributes occuring in c
A_o	Given some set of observation o, $A_o = \{a \mid \langle a, v\rangle \in o\}$, i.e., the set of all attributes occuring in o
$\mathcal{O}$	The set containing all observations
O	A set containing observations, i.e., $O \subseteq \mathcal{O}$
Obs	The set of observations corresponding to the object which needs to be classified
obs_c	The set of observations corresponding to the class c
a, b, o	An observation, usually with some indice indicating its value
L	A finite language for building sentences
S	A subset of L used as parameter
t	Denoting the propositional letter which is always mapped into 1
f	Denoting the propositional letter which is always mapped into 0
$[T]_1^S$	The formula T in which all occurrences (both positive and negative) of letters belonging to $L \setminus S$ are replaced by f.
$[T]_3^S$	The formula T in which all occurrences (both positive and negative) of letters belonging to $L \setminus S$ are replaced by t.

$T \vdash_1^S \phi$	Holds iff every S-1-interpretation (Definition 9.2.0.5) that satisfies T also satisfies ϕ.
$T \vdash_3^S \phi$	Holds iff every S-3-interpretation (Definition 9.2.0.4) that satisfies T also satisfies ϕ.
S_W	The solution set of classes that satisfy the weak classification criterion
S_S	The solution set of classes that satisfy the strong classification criterion
S_E	The solution set of classes that satisfy the explanative classification criterion
S_{W1}^S	The solution set of classes that satisfy the weak classification criterion in which every $\vdash$ is replaced by $\vdash_1^S$
S_{W3}^S	The solution set of classes that satisfy the weak classification criterion in which every $\vdash$ is replaced by $\vdash_3^S$
S_{S1}^S	The solution set of classes that satisfy the strong classification criterion in which every $\vdash$ is replaced by $\vdash_1^S$
S_{S3}^S	The solution set of classes that satisfy the strong classification criterion in which every $\vdash$ is replaced by $\vdash_3^S$
S_{E1}^S	The solution set of classes that satisfy the explanative classification criterion in which every $\vdash$ is replaced by $\vdash_1^S$
S_{E3}^S	The solution set of classes that satisfy the explanative classification criterion in which every $\vdash$ is replaced by $\vdash_3^S$

Bibliography

[Boddy and Dean, 1989] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings IJCAI–89*, Detroit, Michigan USA, August 1989.

[Cadoli, 1993] Marco Cadoli. A survey of complexity results for planning. In A. Cesta and S. Gaglio, editors, *Italian Planning Workshop*, pages 131–145, Rome, Italy, 1993.

[Cadoli, 1996] M. Cadoli. Panel on "Knowledge Compilation and Approximation": Terminology, Questions, and References. *Fourth International Symposium on Artificial Intelligence and Mathematics (AI/MATH-96)*, pages 183–186, 1996.

[Cadoli and Donini, 1997] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications-The European Journal for Artificial Intelligence*, 10:137–150, 1997. Printed in 1998.

[Cadoli and Schaerf, 1995] Marco Cadoli and Marco Schaerf. Approximate inference in default reasoning and circumscription. *Fundamenta Informaticae*, 23:123–143, 1995.

[Cadoli *et al.*, 1994] Marco Cadoli, Francesco M. Donini, and Marco Schaerf. Is intractability of non-monotonic reasoning a real drawback? In *National Conference on Artificial Intelligence*, pages 946–951, 1994.

[Chandra and Markowsky, 1978] A. K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24:7–11, 1978.

[Chopra *et al.*, 2001] Samir Chopra, Rohit Parikh, and Renata Wassermann. Approximate belief revision. *Logic Journal of the IGPL*, 9(6):755–768, 2001.

[Cohen, 1995] P. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[Cook, 1971] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computation*, pages 151–158, 1971.

[Dalal, 1992] Mukesh Dalal. Efficient propositional constraint propagation. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 409–414, San Jose, California, 1992. American Association for Artificial Intelligence.

[Dalal, 1996a] M. Dalal. Semantics of an anytime family of reasoners. In W. Wahlster, editor, *Proceedings of ECAI-96*, pages 360–364, Budapest, Hungary, August 1996. John Wiley & Sons LTD.

[Dalal, 1996b] Mukesh Dalal. Anytime families of tractable propositional reasoners. In *International Symposium on Artifical Intelligence and Mathematics AI/MATH-96*, pages 42–45, 1996. Extended version submitted to Annals of Mathematics and Artifical Intelligence.

[Darwiche and Marquis, 2001] Adnan Darwiche and Pierre Marquis. A perspective on knowledge compilation. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, Washington, USA, August 4th-10th 2001.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the seventh National conference on artificial intelligence AAAI–88*, pages 49–54, Saint Paul, Minnesota, 1988.

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directional Resolution: The Davis-Putnam procedure, revisited. In *Proceedings of the Fourth International Conference on Knowledge Representation and Reasoning (KR-94)*, pages 134–145. Morgan Kaufmann, 1994.

[deKleer, 1990] J. de Kleer. Exploiting locality in a tms. In *Proceedings of the Eight National Conference on Artificial Intelligence (AAAI-90)*, pages 264–271, 1990.

[deKleer, 1992] J. de Kleer. An improved algorithm for generating prime implicates. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 780–785, 1992.

[delVal, 1994] A. del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 551–561, 1994.

[delVal, 1995] A. del Val. An Analysis of Approximate Knowledge Compilation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 830–836, 1995.

[delVal, 1999] A. del Val. A new method for consequence finding and compilation in restricted languages. In *Proceedings of the sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 259–264, 1999.

[Dowling and Gallier, 1984] W. P. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.

[Ernst *et al.*, 1997] M. D. Ernst, T. D. Millstein, and D. S. Weld. Automatic SAT-Compilation of Planning Problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1169–1176, 1997.

[Erol *et al.*, 1995] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. In *Artificial Intelligence*, volume 76, pages 75–88, 1995. (detailed proofs are given in the University of Maryland Technical Report CS-TR-2797 (also listed as UMIACSTR-91-154 and SRC-91-96)).

[Fensel *et al.*, 2003] Dieter Fensel, James Hendler, Henry Lieberman, and Wolfgang Wahlster, editors. *Spinning the semantic web: bringing the World Wide Web to its full potential*. The MIT Press, February 2003.

[Fikes *et al.*, 1971] R. E. Fikes, , and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Ghallab *et al.*, 1998] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl — the planning domain definition language, 1998.

[Haas, 1987] Andrew Haas. The case for domain-specific frame axioms. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence, Proceedings of the 1987 Workshop*. Morgan Kaufmann, 1987.

[Hayes-Roth, 1984] F. Hayes-Roth. Knowledge-based expert systems — the state of the art in the US. In J. Fox, editor, *Expert Systems: state of the art report*. Pergamon Infotech, Oxford, 1984.

[Horvitz, 1987] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In L. N. Kanal, T. S. Levitt, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*, pages 301–324. Elsevier, Amsterdam, The Netherlands, 1987.

[IEEE, 1990] IEEE. IEEE standard glossary of software engineering terminology, 1990. IEEE Standard 610.12-1990, ISBN 1-55937-067-X.

[Jackson and Pais, 1990] P. Jackson and J. Pais. Computing prime implicants. In *Proceedings of the Tenth International Conference on Automated Deduction (CADE-90)*, pages 543–557, 1990.

[Jansen, 2003] M. G. Jansen. *Formal explorations of knowledge intensive tasks*. PhD thesis, University of Amsteram, June 2003.

[Jansen *et al.*, 2000] M. G. Jansen, A. Th. Schreiber, and B. J. Wielinga. Adapting tableaux for classification. In *Knowledge Engineering and Knowledge Management: 12th International Conference (EKAW)*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 334–351, Juan-les-Pins, 2000. Berlin/Heidelberg, Springer Verlag.

[Kautz and Selman, 1991] H. Kautz and B. Selman. A general framework for knowledge compilation. In *Proceedings of the International Workshop on Processing Declarative Knowledge (PDK)*, 1991.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 359–363, 1992.

[Kautz and Selman, 1994] H. Kautz and B. Selman. An Empirical Evaluation of Knowledge Compilation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 155–160, 1994.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, 1996.

[Kautz and Selman, 1998] H. Kautz and B. Selman. The Role of Domain-Specific Knowledge in the Planning as Satisfiability Framework. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, Pittsburgh, 1998.

[Levesque, 1984] H. J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the Fourth National Conference on Artificial Intelligence (AAAI-84)*, pages 198–202, 1984.

[Levesque, 1988] H. J. Levesque. Logic and the complexity of reasoning. *Journal of Philosophical Logic*, 17:355–389, 1988.

[Levesque, 1989] H. J. Levesque. A knowledge-level account of abduction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 1061–1067, 1989.

[Liberatore, 1998] Paolo Liberatore. On the compilability of diagnosis, planning, reasoning about actions, belief revision, etc. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 144–155, 1998.

[Marquis, 1995] P. Marquis. Knowledge compilation using theory prime implicates. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 837–843, 1995.

[Marquis and Sadaoui, 1996] P. Marquis and S. Sadaoui. A new algorithm for computing theory prime implicates compilations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 504–509, 1996.

[McAllester, 1990] David McAllester. Truth maintenance. In *Proceedings AAAI90*, pages 1109–1116. Morgan Kaufmann Publishers, 1990. internet file ftp.ai.mit.edu:/pub/dam/aaai90.ps.

[McCarthy, 1959] J. McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanisation of Thought Processes*, pages 75–91, London, 1959. Her Majesty Stationary Office, London. Reprinted in *Formalizing common sense.* V. Lifschitz, editor. Ablex Publishing Corporation.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502, Edinburgh, 1969. University Press.

[McDermott, 1981] D. McDermott. Artificial intelligence meets natural stupidity. In J. Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 143–160. MIT Press, Cambridge, MA, 1981.

[Menzies and vanHarmelen, 1999] Tim Menzies and Frank van Harmelen. Evaluating Knowledge-Engineering Techniques. *International Journal of Human-Computer Studies*, 51(4):715–727, October 1999.

[Minato, 1993] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th ACM/IEEE Design Aautomation Conference*, pages 472–277, 1993.

[Mitchell *et al.*, 1992] David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions for SAT problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, Menlo Park, California, 1992. AAAI Press.

[Preece *et al.*, 1997] A. D. Preece, S. Talbot, and L. Vignollet. Evaluation of verification tools for knowledge-based systems. *International Journal of Human-Computer Studies*, 47:629–558, 1997.

[Puppe *et al.*, 1994] F. Puppe, K. Poeck, U. Gappa, S. Bamberger, and K. Goos. Wiederverwendbare Bausteine für eine konfigurierbare Diagnostik-shell. *Künstliche Intelligenz*, 94(2):13–18, 1994.

[Puppe *et al.*, 1996] F. Puppe, U. gappa, K. Poeck, and S. Bamberger. *Wissensbasierte Diagnose- und Informationssysteme*. Springer-Verlag, Juli 1996.

[Quine, 1959] W. V. O. Quine. On cores and prime implicants of truth functions. *American Mathematical Monthly*, 66, 1959.

[Reiter and deKleer, 1987] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, pages 183–188, 1987.

[Russell and Zilberstein, 1991] S. J. Russell and S. Zilberstein. Composing real-time systems. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 212–217, Sydney, Australia, 1991.

[Salton and McGill, 1983] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[Schaerf and Cadoli, 1995] Marco Schaerf and Marco Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.

[Schrag and Crawford, 1996a] R. Schrag and J. Crawford. Compilation for critically constrained knowledge bases. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 510–515, 1996.

[Schrag and Crawford, 1996b] Robert Schrag and James M. Crawford. Implicates and prime implicates in random 3SAT. *Artificial Intelligence*, 81(1-2):199–222, 1996.

[Schumann and Fischer, 1997] J. Schumann and B. Fischer. NORA/HAMMR: Making deduction-based software component retrieval practical. In *Automated Software Engineering (ASE)'97*, pages 246–254. IEEE, 1997.

[Selman and Kautz, 1991] B. Selman and H. A. Kautz. Knowledge compilation using Horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 904–909, 1991.

[Selman and Kautz, 1996] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43:193–224, 1996.

[Shadbolt *et al.*, 1999] Nigel Shadbolt, Kieron O'Hara, and Louise Crow. The experimental evaluation of knowledge acquisition techniques and methods: history, problems and new directions. *International Journal of Human-Computer Studies*, 51(4):729–755, October 1999.

[Simon and delVal, 2001] L. Simon and A. del Val. Efficient consequence finding. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[Stefik, 1993] M. Stefik. *Introduction to Knowledge Systems*. Los Altos, California. Morgan Kaufmann, 1993.

[tenTeije and vanHarmelen, 1996] A. ten Teije and F. van Harmelen. Computing approximate diagnoses by using approximate entailment. In G. Aiello and J. Doyle, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, Boston, Massachusetts, November 1996. Morgan Kaufman.

[tenTeije and vanHarmelen, 1997] A. ten Teije and F. van Harmelen. Exploiting domain knowledge for approximate diagnosis. In M.E. Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, volume 1, pages 454–459, Nagoya, Japan, August 1997. Morgan Kaufmann.

[Tison, 1967] P. Tison. Generalized consensus theory and application to the minimization of boolean circuits. *IEEE Transactions on Computers*, EC-16:446–456, 1967.

[vanGelder and Tsuji, 1996] A. van Gelder and Y. K. Tsuji. Satisfiability testing with more reasoning and less guessing. In D. S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisifibility: Second DIMACS Implementation challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1996.

[vanHarmelen and tenTeije, 1995] F. van Harmelen and A. ten Teije. Approximations in diagnosis: motivations and techniques. In C. Bioch and Y.H. Tan, editors, *Proceedings of the Dutch Conference on AI (NAIC'95)*, Rotterdam, June 1995.

[vanHarmelen and tenTeije, 1998] F. van Harmelen and A. ten Teije. Characterising approximate problem-solving by partial pre- and postconditions. In *Proceedings of ECAI'98*, pages 78–82, Brighton, August 1998.

[Wielinga *et al.*, 1998] B. J. Wielinga, J. M. Akkermans, and A. Th. Schreiber. A competence theory approach to problem-solving method construction. *International journal of Human Computer Studies*, 49:315–338, 1998.

[Zilberstein, 1993] S. Zilberstein. *Operational rationality through compilation of anytime algorithms*. PhD thesis, Computer science division, university of California at Berkley, 1993.

[Zilberstein, 1996] S. Zilberstein. Using anytime algorithms in intelligent systems. *Artificial Intelligence Magazine*, fall:73–83, 1996.

[Zilberstein and Russell, 1995] S. Zilberstein and S. J. Russell. Approximate Reasoning Using Anytime Algorithms. In S. Natarajan, editor, *Imprecise and Approximate Computation*. Kluwer Academic Publishers, 1995.

[Zilberstein and Russell, 1996] Shlomo Zilberstein and Stuart J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.

Samenvatting

Dit proefschrift heeft de volgende doelstelling:

> Een theoretische en empirische analyse van approximatie in symbolische probleemoplosmethoden.

Approximatie is een nuttige aanpak voor het oplossen van complexe problemen. In plaats van een exacte oplossing voor een probleem gaan we op zoek naar een oplossing die de exacte oplossing benadert. Er zijn verscheidene redenen te geven waarom een benaderende oplossing te prefereren valt voor een exacte oplossing. We bespreken twee van deze redenen hier in meer detail.

Ten eerste, kan approximatie gebruikt worden om een afweging te maken tussen rekentijd en kwaliteit van de oplossing. Enerzijds, zijn sommige problemen zo complex dat ze onmogelijk binnen een redelijke tijd opgelost kunnen worden. Anderzijds, moeten sommige problemen binnen een bepaalde deadline opgelost worden. Deze deadline hoeft niet altijd van tevoren bekend te zijn.

Ten tweede, kan approximatie gebruikt worden om een probleemoplosmethode robuuster te maken tegen incorrecte en incomplete data. Door de invoer van incorrecte en/of incomplete data kan het zijn dat een probleem te weinig of teveel oplossingen heeft. Door bijvoorbeeld de eisen van het probleem te verzwakken of te versterken, krijgen we een nieuwe formalisering die wel een acceptabel aantal oplossingen heeft. Deze oplossingen zijn dan benaderende oplossingen van het originele probleem.

In dit proefschrift beperken we ons tot symbolische probleemoplosmethoden. Karakteristiek voor deze methoden is het gebruik van een logica voor de representatie en logische deductie voor de inferentie. Aangezien dit geen numerieke problemen zijn, is er geen duidelijke afstandsfunctie die aangeeft 'hoe ver' een benadering verwijderd is van de optimale oplossing. Kortom, het gebruik van approximatie in symbolische probleemoplosmethoden is niet vanzelfsprekend.

Bovenstaande licht de doelstelling van dit proefschrift toe, maar de formalisering van de doelstelling is nog zeer algemeen. In dit proefschrift richten we ons daarom op drie specifieke problemen die vallen binnen de gegeven doelstelling. Deze worden hieronder in meer detail toegelicht.

I. Kwantitatieve maten voor Kennissystemen

Kennissystemen zijn systemen die de kennis bevatten van één of meerdere experts voor het oplossen van een specifieke taak (dokter, monteur, etc.). Kennissystemen zijn vaak goed in staat om incorrecte en/of incomplete data te verwerken. Uit verificaties van Kennissystemen is gebleken dat zelfs Kennissystemen die fouten bevatten op een acceptabel niveau functioneren. De robuustheid van Kennissystemen t.o.v. incorrecte en incomplete data wordt beschouwd als een belangrijk onderdeel in de validatie van Kennissystemen.

Huidig onderzoek naar de robuustheid van Kennissystemen is echter beperkt tot praktijkervaringen en kwalitatieve analyses. In dit proefschrift laten we zien dat het zowel mogelijk als nuttig is om ook kwantitatief onderzoek te verrichten naar de robuustheid van Kennissystemen. Allereerst wordt er een methode voorgesteld om de robuustheid van Kennissystemen te kwantificeren. Vervolgens wordt deze methode daadwerkelijk toegepast op een Kennissysteem.

De methode om de robuustheid van Kennissystemen te kwantificeren is gebaseerd op zogeheten ‘degradatie studies’. Dit betekent dat de kwaliteit van de invoer van het Kennissysteem systematisch wordt verminderd en dat daarbij gelet wordt op de veranderingen in de kwaliteit van de uitvoer van het Kennissysteem. De kwaliteitsmaat voor de invoer hangt af van het specifieke systeem dat bestudeerd wordt, maar vaak kan incompleetheid of incorrectheid van de invoer vertaald worden naar een concrete kwaliteitsmaat. De kwaliteitsmaat voor de uitvoer wordt in de voorgestelde methode bepaald door de maten recall en precision, die wel bekend zijn in het vakgebied Information Retrieval.

De methode om de robuustheid van Kennissystemen te kwantificeren wordt tevens toegepast op een Kennissysteem. Dit Kennissysteem wordt gebruikt voor het classificeren van veel voorkomende planten in het zuiden van Duitsland. Met deze experimentele analyse wordt aangetoond dat de voorgestelde methode daadwerkelijk uitgevoerd kan worden en bovendien kan resulteren in, soms verrassende, inzichten in het geanalyseerde Kennissysteem.

II. Kenniscompilatie

Veel systemen maken gebruik van een logica voor de representatie. Het gebruik van logica heeft een aantal voordelen, maar heeft echter ook een belangrijk nadeel: de computationele complexiteit van inferentie. Veel technieken zijn in de loop der jaren aangedragen om dit probleem op te lossen. Kenniscompilatie is zo’n techniek.

Het idee achter kenniscompilatie is dat veel problemen conceptueel te splitsen zijn in twee delen, nl. een vast deel en een variabel deel. Het vaste deel blijft constant over meerdere probleem instanties, terwijl het variabele deel varieert over meerdere probleem instanties. Omdat het vaste deel constant blijft, kan deze vertaald worden, zodat de vertaling betere computationele eigenschappen heeft. Uiteraard is de vertaling zelf computationeel complex, maar deze hoeft slechts één keer uitgevoerd te worden, kan al gedaan worden voordat er daadwerkelijk problemen opgelost gaan worden, en kan hergebruikt worden voor het oplossen van meerdere probleem instanties.

Hoewel kenniscompilatie interessant is vanuit een theoretisch perspectief, is er nog weinig bekend over succesvolle toepassingen. In dit proefschrift richten we ons op planningsproblemen. Planning kan vertaald worden in propositie logica, is computationeel complex, en is te schrijven als een vast deel (domein, acties) en een variabel deel (begin- en eindtoestand). Planning is dus een kandidaat voor kenniscompilatie. We voeren een experimentele analyse uit van kenniscompilatie toegepast op planning.

III. Benaderende Classificatie

Dit deel van het proefschrift geeft aan hoe een gestructureerde analyse uitgevoerd kan worden van een benaderende consequentie relatie d.m.v. een combinatie van theoretisch en experimenteel onderzoek. We maken hierbij gebruik van een algemene benaderingsmethode ontwikkeld door Cadoli en Schaerf en passen deze toe om benaderende classificatie vormen te verkrijgen.

Als eerste geven we, door gebruik te maken van een verzameling theoretisch raamwerk voor klassieke classificatie, een formalisatie van classificatie in propositie logica.

Vervolgens passen we de benaderende consequentie operator toe op de formalisatie in propositie logica. Dit resulteert in een aantal benaderende classificatie vormen. D.m.v. een theoretische analyse leiden we een aantal stellingen af die de benaderende classificatie vormen karakteriseren.

Tenslotte voeren we een experimentele analyse uit om meer inzicht te krijgen in het incrementele benaderingsgedrag van de benaderende classificatie vormen. In het bijzonder, geven we een aantal heuristieken voor een parameter van de benaderingsmethode van Cadoli en Schaerf en analyseren we het bijbehorende benaderingsgedrag.

Index

SIKS Dissertatiereeks

1998

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation

2000

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE) *Prototyping of CMS Storage Management*

2000-3 Carolien M.T. Metselaar (UvA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectie

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coupé (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VU)
Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel (KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (UvA)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UvA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UvA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

2004

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business