1994

An approach to commonsense reasoning: Default logic augmented with certainty factors.

Viren. Parasram
University of Windsor

An Approach to Commonsense Reasoning : Default Logic Augmented with Certainty Factors

by

Viren Parasram

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree of
Master of Science at the
University of Windsor
Windsor, Ontario,
Canada
1994

ISBN 0-612-01418-5

VIREN PARASRAM

Name An Approach to Commonsense Reasoning : Default Logic Augmented with Certainty Factor

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

COMPUTER SCIENCE | 0 9 8 4 | U·M·I

SUBJECT TERM | SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523
Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575
Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984
Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

Abstract

An active area of research is to find a logic which models human commonsense reasoning. Several non-monotonic logics have been proposed. However, each has deficiencies associated with it. Default Logic is one such non-monotonic formalism.

We examine some of the problems with Default Logic as pointed out by *[Nut83, Nut87, Isr80, Pea90]* and present an approach which addresses some of these deficiencies.

In our approach, we attach a certainty factor to each statement in the knowledge base and use this information to give the user the most certain answer to a query.

This gives us a certainty factor default logic (CFDL). We present the syntax semantics and proof theory for this logic.

A resolution algorithm has been developed and implemented. This implementation shows that the proposed logic works with some benchmark examples *[Lif89]* for non-monotonic reasoning. It can also be used as a testbed for different calculi of uncertainty.

To Adita

Acknowledgments

I would like to acknowledge the guidance and support provided by Dr. Joan Morrissey. She has been very patient and provided me with invaluable advice, suggestions and comments. I would also like to thank Dr. Richard A. Frost and Dr. Subir Bandyopadhyay for their advice and encouragement. I am grateful to Dr. Robert Pinto and Dr. Liwu Li for their roles in my supervisory committee. Finally, I would like to thank Stephen Karamatos and Walid Mnaymneh for all their help throughout the various stages of my work.

TABLE OF CONTENTS

List of Figures

Chapter 1: INTRODUCTION

1 Our Area of Research

In this thesis we are interested in approaches to commonsense reasoning. For example, if we are told "***most birds fly***" and "***Tweety is a bird***", we may use commonsense reasoning to conclude "***Tweety flies***". However the conclusion "***Tweety flies***" is not certain. There are many different scenarios which would refute "***Tweety flies***". For example, Tweety may be a penguin or Tweety may be injured. Despite these possibilities the conclusion "***Tweety flies***", based on the two pieces of information given, can be said to be an intelligent conclusion.

Intelligent behavior rests in the ability to reason in the absence of complete and certain information. Life by its very nature is uncertain. We drive to work believing that we have a job, the office still exists, that the car will get us there, but none of this is completely certain. Our information is incomplete but we are not paralysed into inaction. We reason, make intelligent decisions and go about our lives based on the available incomplete uncertain information.

Non-monotonic formalisms are frequently used for commonsense reasoning since they are more flexible. Default Logic is one such formalism. Default logic models the human commonsense reasoning process of jumping to conclusions, based on prototypical or default rules, when information is incomplete. The logic contains default rules which are used to make intelligent guesses when the information available is incomplete. For example, given the information "***John is a Canadian***" and the default rule "***most Canadians play hockey***" then we can reasonably conclude "***John plays hockey***". Of course, this is only a guess. We may find out later that John hates hockey and therefore never plays it. However, the default rule allows us to make a reasonably good guess in the absence of complete information.

2 The Problems Addressed in this Thesis

Default Logic is one non-monotonic formalism used for commonsense reasoning, but there are some deficiencies with it. Two problems are specifically of interest in this thesis.

Firstly, Default Logic does not offer any suggestions when two default rules are in conflict. For example, we have the following information:

```
Most Republicans are hawks  (default rule)
Most Quakers are doves      (default rule)
     Nixon is a Republican
     Nixon is a Quaker
```

The two defaults are in conflict and DL cannot tell us if Nixon is a hawk or a dove.

Secondly, Default Logic does not distinguish between "**the statement P is true**" and "**there is reason to believe P is true**". For example, given the information:

```
Typically birds fly
Peter is a bird
Paul flies
```

then clearly we can conclude "***Peter flies***". However, Default Logic makes no distinction between the statement "***Peter flies***" and the statement "***Paul flies***", but there is clearly a difference since the former is derived via a default rule and as such is only a "**best guess**". It is not known for certain that "***Peter flies***".

This thesis address these two specific problems.

3 Our Solution

To solve these two problems we propose to attach a certainty factor to each item of information in the knowledge base. We develop a logic called certainty factor default logic (CFDL). This is essentially Default Logic augmented with certainty factors. In the thesis we show how this approach solves some problems and has some other advantages also.

3.1 Thesis Statement

Default logic is an approach to NMR, but it is deficient in some respects. Specifically of concern here is the fact that it can not handle conflicting defaults and it does not distinguish between the statements "P is true" and "there is reason to believe P is true". The hypothesis here is that these two problems can be addressed by attaching a certainty factor to each statement in the KB which represents the certainty that the statement is true or false.

4 Thesis Outline

Chapter 2 provides background for our research area. We discuss some of the basic concepts such as monotonic and non-monotonic reasoning. We examine the criticisms of non-monotonic reasoning. We look at Default Logic and give an example of a system which implements DL. We then examine some methods for handling uncertainty.

Chapter 3 examines in detail the problems associated with Default Logic. In Chapter 4 we outline our solution to the problem and give examples to illustrate how the solution works.

Chapter 5 presents our proposed certainty factor default logic. We give details of the syntax, semantics and proof theory. Chapter 6 gives the implementation details of a system based on our logic. We outline the resolution strategy, data structures and resolution algorithm.

Chapter 7 has a discussion of our findings and some future work. We also give some of our conclusions and show how these findings support our thesis statement.

Appendix A is a list of abbreviations. Appendix B is a set of definitions for the subject area. Appendix C is a survey of non-monotonic reasoning and appendix D is a bibliography of non-monotonic reasoning. Appendix E gives the source code for our implementation.

Chapter 2: BACKGROUND

1 Introduction

In this chapter we give background information on our research area. We describe what is meant by monotonic reasoning (MR) and non-monotonic reasoning* (NMR). We show how NMR differs from MR and why we need NMR. We also look at some criticisms of NMR, give a brief overview of default logic (DL) and examine some methods of handling uncertainty. (A complete survey of NMR is given in Appendix C)

2 What is Monotonic Reasoning ?

If the addition of new information to the knowledge base does not block the derivation of a previously derivable conclusion then our reasoning is monotonic. Hence, given a set of information P, then if we can conclude q from P, then we must also be able to conclude q if new information is added to P. For example, if given "***Birds fly***" and "***Tweety is a bird***" we can conclude "***Tweety flies***". If on addition of the fact "***Penguins do not fly***" we are still able to conclude "***Tweety flies***", then our reasoning is monotonic.

In monotonic reasoning the number of conclusions must increase monotonically with the amount of information available. This follows since no old conclusion can be made false and therefore the number of possible conclusions can only increase. More importantly, from our perspective, is the fact that the number of conclusions can never decrease in monotonic reasoning.

* For formal definitions of NMR and MR see survey pg C.84

Monotonic reasoning is clearly inappropriate for commonsense reasoning. Recall the Tweety example where our present information is "***Birds fly***", "***Penguins do not fly***" and "***Tweety is a bird***". If we were to add the information that "***Tweety is a penguin***" then we can conclude "***Tweety flies***" since Tweety is a bird. We can also conclude "***Tweety does not fly***" since Tweety is a penguin. Tweety can not both "**fly**" and "**not fly**". However if we are reasoning monotonically we can derive both these conclusions and there is a conflict.

The problem pivots on the fact that we want to say "***most birds fly***" which is not possible in classical logic. "***Most birds fly***" implies that there are some birds that do not fly. Thus, the conclusion "***Tweety flies***" from the information "***Most birds fly***" and "***Tweety is a bird***" can not be made with absolute certainty. In order that our conclusion "***Tweety flies***" be absolutely certain we would have to know what exactly is meant by most in the statement "***Most birds fly***". In other words we would need to have complete information on the meaning of "***Most***". We would need to know all the exceptions to "***most birds fly***" and also that Tweety is not an exception.

For example, we would have to say "***Birds fly if they are not penguins or emus or have their wings clipped or are dead or . . .*** ". We can never be certain that our list is complete. For monotonic reasoning it would have to be complete. In real life our information can never be absolutely certain. Despite this lack of complete and certain information we continue to make decisions which seem to be the best at the time and simply change them, if necessary, when new information comes to light. A key feature of NMR is that it allows old conclusions to be revised in the light of new knowledge.

3 What is Non-monotonic Reasoning ?

Reasoning is non-monotonic if the conclusions may be revised when new information comes to light. For example, given:

```
Most birds fly
Penguins do not fly
Tweety is a bird
```

we can reason "***Tweety flies***". However, if we find out "***Tweety is a penguin***" we now have to conclude "***Tweety does not fly***". Our previous conclusion is no longer true, it has been retracted and a new conclusion is added.

There are different approaches to NMR. One approach uses a closed world assumption (CWA). In this approach any fact that is not specified or derivable from information specified is assumed to be false. Thus in the CWA, there is no need to explicitly represent negative information. The formal approaches to NMR which use the CWA, such as Circumscription *[McC80]*, use the CWA to define a possible world which will satisfy the incomplete information. For example, given

```
A is a block and B is a block,
```

then on circumscription we get

```
if x is a block it must be A or B
otherwise it is not a block.
```

One problem here is that we are unable to differentiate between certain and possible inferences such as

```
C is not a block and
```

```
C is possibly a block.
```

The other approaches to NMR are the non-closed world. They utilize different strategies to complete the KB with the incomplete information. One such approach is Default Logic. It adds rules, known as default rules, to the KB, which are used when there is no specific information. For example, if

```
A and B are blocks and
C is not known to be a block.
```

In DL, we can add the default rule "***Most objects are not blocks***". We can use this rule to conclude C is not a block. Thus, default rules give a choice when there is no conclusive information. However, one problem is that we still unable to distinguish between answers that are certain and those those that are derived from default rules.

Some of the non-closed world approaches are Numeric. In the Numeric approaches, the problem of incomplete information is addressed by assigning different probabilities, certainty factors or belief measures to the information. The problem with some of these approaches is the need for prior information and relationships between the information in the KB. Traditionally, Numeric approaches have been regarded as suitable for domains in which such prior information is available. However, in more recent times a more subjective approach to probabilities has made it possible to use the numeric approaches even when prior information is unavailable (See survey of NMR in Appendix C).

In our research we look at ways in which we can merge the desirable aspects of the Numeric and Non-Numeric approaches to NMR. More specifically we look at merging DL with certainty factors.

The possible application areas for computers will be endless if they are capable of commonsense reasoning. They are many ways in which this problem can be addressed. NMR is one possible approach to the problem. We focus our research on NMR despite the criticisms this approach has attracted.

4 Critics of Approaches to NMR

There are several criticisms of the approaches to NMR. One *[Pea90]* criticism is that the formalisms do not build on traditional logics. Another criticism, *[Pea90]* concerns the problem of consistency checking, inherent in NMR, which is very difficult. Yet another criticism *[Nut83]* is that the formalisms do not make provisions for conclusions which are not completely certain. We now look more closely at these criticisms.

4.1 NMR should build on Traditional logics

One criticism *[Pea90]* is that NMR formalisms attempt to define a new non-monotonic logic rather than extending existing approaches to reasoning. The argument is that existing approaches to reasoning are well-defined and well-understood. Hence, we should look at ways in which these approaches can be augmented to allow NMR rather than look for a new non-monotonic logic or new formalisms which may not be consistent with traditional approaches. An example of a formalism which takes this approach is Circumscription where first order logic is augmented with a form of NMR.

The probabilists *[Pea90]* similarly argue that an approach to commonsense reasoning based on probability theory should be pursued instead of NMR since probability theory is much better understood than the non-numeric approaches.

4.2 Consistency checking is very difficult

Israel *[Isr80]* argues that the formal approaches to NMR are in general, non semi-decidable. This means that if a conclusion follows from the available information it may not always be possible to prove that this is the case. Israel argues that NMR systems are very slow because of the repeated need for consistency checking.

In reasoning we make conclusions that follow from the available information. These conclusions must be consistent with the available information. The process of ensuring that our information is consistent is known as consistency checking.

In monotonic reasoning the information given is consistent. If we add a new piece of information "***p***" then to ensure that **p** and our old information is consistent, we need only show that "***not p***" is not a possible conclusion. For example, the information

```
Tweety is a bird
Birds fly
Penguins don't fly
```

is consistent. However, adding "***Tweety is a penguin***" makes the KB inconsistent since we can conclude "***Tweety flies***" (**p**) and "***Tweety does not fly***" (**not p**).

The problem with consistency checking in NMR is that the addition of a new piece of information "**p**" may result in inconsistencies other than "**not p**". Hence a proof that "**not p**" is not derivable is insufficient in this case. For example, in the Tweety example, when we added the fact "***Tweety is a penguin***" (**p**) the inconsistency did not arise from "***Tweety is not a penguin***" (**not p**) but rather from "***Tweety flies***" (q) and "***Tweety does not fly***" (**not q**).

Consistency checking in NMR will require a check to ensure that no pair of possible conclusions are in conflict. This is a very difficult task even for a small KB.

We agree that the process of consistency checking in NMR is very difficult. However, this is a problem with all reasoning formalisms where the information may be incomplete.

4.3 Only true/false conclusions

It can be argued that many uncertain conclusions are made when reasoning with incomplete information. For instance, in the Tweety example the conclusion "***Tweety flies***" is not certain. However present approaches to NMR do not allow conclusions of the form "***There is reason to believe that Tweety flies***".

Nutter *[Nut83]* notes that non-monotonic approaches do not distinguish between guarded statements of the form "**there is reason to suppose P**" and statements of the form "**P**" and hence no distinction is made between conclusions which the premises warrant without reservation and those which the premises only suggest.

We find this criticism of Nutter to be very interesting. One objective of our research is to overcome these shortcomings.

Despite these criticisms *[Nut83, Pea90]* research in NMR is very active in both the theoretical aspects and practical applications. (For further details see Appendix C)

5 Summary: Default Logic

Default reasoning is an approach to reasoning with incomplete information where, in the absence of certain information, one makes conclusions based on what is "*normally*" expected.

Reiter's Default Logic *[Rei80]* uses a default approach to NMR and is proposed as a logic for default reasoning. DL provides a representation for commonsense facts of the form "***Most A's are B's***" where "**most**" is interpreted in the prototypical and not in a statistical sense. For example,

```
Most elephants are grey and
Most birds fly
```

are characteristic information for a prototypical or normal elephant and bird respectively. It is not meant for purely statistical information of the type "***Most voters prefer Clinton***" since this does not mean that typically voters prefer Clinton or normal voters prefer Clinton. DL also explains the process of correct reasoning when such defaults are present.

DL consists of default rules which in general have the form

$$\frac{a(X) : b(X)}{c(X)} \tag{1}$$

which is normally interpreted as " **if for some specific X, a(X) is true and it is consistent that b(X) is true then conclude c(X)**". A subset of all defaults have the form

$$\frac{a(X) : b(X)}{b(X)} \tag{2}$$

These are referred to as normal defaults. Using the same syntax, the default "***Most birds fly***" would be represented as

$$\frac{bird(X) : flies(X)}{flies(X)} \tag{3}$$

which is interpreted as "**if X is a bird and it is consistent that X flies then X flies**".

The basic strategy in DL if asked "**Who flies**" will be to search the certain facts (Reiter's "**hard**" facts) for those objects known to fly and if none is found then to use default rules to find a answer. If there is no certain fact or applicable default then a "**no**" answer is given.

In *[Par94]*, DL is classified as a non-closed world non-numeric approach. We classify DL as non-closed world since it uses a non-uniform completion strategy, in contrast to the uniform completion strategy of closed world approaches. We illustrate this point with an example.

Using a CWA, if we can not prove Tweety flies we will assume that Tweety does not fly. However, in DL we can change the completion strategy by using the default "***typically birds fly***" to conclude Tweety flies.

We classify DL as non-numeric since no numeric or statistical information is explicitly represented in the defaults. Reiter *[RC81]* notes that his defaults like "***Most birds fly***" have both a statistical connotation "***The majority of birds fly***" and a sense that a prototypical or normal bird is being described. We found this point significant and, in our research, look at the possible use of some form of statistical information with Reiter's defaults.

In common with all approaches to NMR, DL has some deficiencies. These will be examined in Chapter 3.

6 An Example System: Theorist

In this section, we examine the approach taken by Theorist *[Poo92]* to default reasoning since our system also models default reasoning. This enables us to look at some of the differences between the systems.

Theorist *[Poo88]* is a logical reasoning system for default reasoning and diagnosis. The formulae in the KB are divided into three sets.

- A set of facts which are intended to be true in the world being modelled
- a set of possible hypotheses, which is any ground instance which can be used in an explanation if consistent and
- a set of constraints which are closed formulae used to restrict what can be hypothesized.

In the KB these formulae are represented as:

fact w where w is always true

for example :-

fact bird(tweety)

which means Tweety is a bird.

default d : w where d is a default

for example :-

default birdsfly(X) : flies(X) <- bird(X)

which means there is a default rule named birdsfly which can be used to conclude X flies if X is a bird.

constraint w where w prevents the use of the corresponding default.

```
for example :-
    constraint not birdsfly(X) <- emu(X)
```

which means the default named birdsfly cannot be used if X is an emu.

I will illustrate how these formulae are used to answer queries with an example. The KB is as follows:

```
default birdsfly(X) : flies(X) <- bird(X).
constraint not birdsfly(X) <- not flies(X).
default emusdontfly(X) : not flies(X) <- emu(X).
constraint not emusdontfly(X) <- flies(X).
constraint not birdsfly(X) <- emu(X).
fact bird(X) <- emu(X).
fact bird(X) <- robin(X).
fact bird(tweety).
fact emu(polly).
fact robin(cohen).
fact not flies(tweetum).
fact robin(tweetrob).
```

If the user wants to know who flies then the answers generated would be **Tweety** and **Tweetrob**. The possible answers **Polly** and **Tweetum** are prevented by the constraints.

Theorist answers the query ***Who flies ?*** as follows:

```
if there a fact flies(X) then
   answer yes with the binding to X
else if there is an applicable default
     make the default conclusion provided
     there are no constraints on that named default.
```

The addition of the default ***penguins don't fly*** requires that the constraint ***not birdsfly penguin*** be generated. Thus, in Theorist we are forced to represent all the exceptions to a default explicitly. This is not desirable and our system provides another approach to encode the exceptions to a default (See chapter 4).

7 Methods of Handling Uncertainty

Conclusions made in NMR may not be certain. Hence, it is reasonable to ask: "How certain is a given conclusion ?". Traditionally, numeric methods have attempted to deal with this problem. They assign certainty measures to conclusions. Then they either try to find a model which satisfies all the conclusions or they look for a set of rules which would give the set of conclusions. We look more closely at the rule based systems.

The mechanism for deriving conclusions, in a rule based system, can be implemented using if-then production rules. The problem with these production rules is that they are not independent of each other. Hence, the difficulty lies in the proper propagation of the certainty measures through the proof procedure. Each type of numerical approach uses a different approach to deal with this problem.

Some of the main approaches to this problem are probability theory, certainty theory, Dempster Shafer theory of evidence and Zadeh's fuzzy logic and possibility theory.

7.1 Probability Theory

Probability theory is the classical means of dealing with uncertainty. However, a prior knowledge of the events and their interdependencies is required before an estimate of the various combinations of events can be determined.

For example, if A and B are independent events then, the probability of A and B occurring is given by the product of the probabilities of A and B. That is

P(A and B) = P(A) * P(B).

Similarly we have,

P(A or B) = P(A) + P(B) — P(A and B).

Bayes rule was developed to help estimate probabilities when they are not known. However, a clear knowledge of their interdependencies is still required and this is a problem.

7.2 Certainty Theory

Certainty theory *[SB75]* is, in effect, an approximation to probability theory, but it uses incomplete knowledge *[Mor87]*. The theory was developed in an attempt to model the inexact reasoning processes of medical experts and was implemented as part of the MYCIN expert system *[Sho76]*. The approach taken is to maintain two values for each rule in the system:

- MB[h,e] = X

  is the measure of the increased belief in hypothesis h given evidence e.

- MD[h,e] = Y

is the measure of the increased disbelief in hypothesis h given evidence e. These measures of belief and disbelief correspond to subjective estimates given by experts. They are related to the Probability theory in the following way:

$$MB[h,e] = \left\{ \begin{array}{c} 1\ iff\ P(h) = 1 \\ \frac{max|P(h|e))| - P(h)}{max|1,0| - P(h)}\ otherwise \end{array} \right\} \quad (4)$$

$$MD[h,e] = \left\{ \begin{array}{c} 1\ iff\ P(h) = 0 \\ \frac{min|P(h|e))| - P(h)}{max|1,0| - P(h)}\ otherwise \end{array} \right\} \quad (5)$$

where P(h) is the expert's subjective belief in hypothesis h, and P(h|e) is the experts's subjective belief in hypothesis h given evidence e.

A third measure is the certainty factor which combines the measures of belief and disbelief and is defined as:

CF[h,e] = MB[h,e] — MD[h,e]

The certainty factor is used to rank competing hypotheses. Its advantage is that if exact probabilities cannot be obtained then it can provide a subjective estimate.

7.3 Dempster Shafer Theory of Evidence

This theory *[Sha76]* sets up belief functions over sets of objects. Morrissey *[Mor87]* notes:

"A belief function associates a number between zero and one with a proposition. The number indicates the degree of belief in the proposition given the evidence. The theory concentrates on combining degrees of belief given different bodies of evidence. It is not concerned with how the numbers are determined."

Shafer *[Sha90]* notes:

" The theory of belief functions is based on two ideas: the idea of obtaining degrees of belief for one question from subjective probabilities for a related question and Dempster's rule for combining such degrees of belief when they are based on independent items of evidence."

One problem with Dempster Shafer's theory of evidence is that it is difficult to implement.

The Dempster Shafer Theory of Evidence is not considered further in this work since we are not primarily concerned with belief functions. We do not want to estimate our belief in a particular conclusion but rather estimate our certainty that a conclusion is correct.

7.4 Fuzzy Logic and Possibility Theory

Dubois and Prade *[DP90]* note:

"The expression 'fuzzy logic' is used to refer to a variety of approaches proposing a logical treatment of imperfect knowledge usually referring explicitly to fuzzy-set theory. However, a distinction among these approaches can be made between those that deal primarily with vagueness and those whose primary concern is uncertainty."

Fuzzy logic *[Zad83]* is used to present certain statements which other logics cannot handle. For example, the statement: 'If a car which is being offered for sale is old and cheap then it is probably not in good condition' is rather vague and uncertain and involves a degree of truth *[Mor87]*.

Possibility theory *[Zad78]* is based on earlier work on Fuzzy Set Theory *[Zad65]*. The work in this area has been continued by Dubois and Prade. They discuss a possibilistic logic *[DP90]*, which is a logic of partial ignorance and note that the possibility theory captures, in a very simple way, states of knowledge ranging from complete information to total ignorance.

8 Conclusion

In this chapter we have shown that NMR is needed when knowledge is incomplete. Despite criticisms research in NMR is very active. We have given an overview of DL as an approach to NMR and concluded the chapter by looking at some methods of handling uncertainty.

In the next chapter we examine some of the problems with Default Logic.

Chapter 3: PROBLEMS WITH DEFAULT LOGIC

In this chapter we look at the problems associated with the DL approach to modelling human commonsense reasoning. We describe some of the proposed solutions *[RC81, Bre91]*. We also discuss Nutter's *[Nut83]* criticism of DL.

1 Interacting Defaults

Default rules provide a means of jumping to conclusions. However, these defaults are not always mutually exclusive. They interact with each other and although they give intuitively correct results most of the time, they cannot be guaranteed to do so at all times. Thus, one of the problems with DL is that when defaults interact they do not always lead to intuitively correct conclusions. One way in which default rules interact is when the conclusion of one is the same as the premiss of the other. For example, given the defaults:

```
Elementary students are children
Children are not employed
```

we note the conclusion of the first default (children) is the same as the premiss of the other (Children). Hence, these defaults interact and in this case give the conclusion

```
Elementary students are not employed.
```

In this example, this conclusion is what we would intuitively expect.

Interacting defaults do not always lead to intuitive correct conclusions. For example, given the defaults:

```
High school dropouts are adults
Adults are employed
```

they will interact to give the conclusion

```
High school dropouts are employed.
```

This is not the kind of conclusion we would like to make given the original defaults. It seems more likely that High school dropouts will not be employed.

Also *[RC81]* we have the problem that:

" . . . the general class of default theories is mathematically intractable."

This paper also notes that "*normal*" default theories had the desirable properties (e.g. extensions always exist, a proof theory, conditions for belief revision) only when interactions between default rules are ignored.

If we are to maintain a normal representation for default rules then we must find a way to deal with the interaction between default rules. If we choose a "*non-*

normal" representation then we will have to deal with the loss of the desirable properties of " *normal*" default theories. The problems with DL stem from this conflict.

The problems with DL stem from the interaction of defaults and the need for a "*non-normal*" representation for some defaults.

Interacting defaults are not considered any further in this thesis.

2 Conflict with Default and Certain Rules

Default rules are rules in which the conclusions are most likely, but not certain, to follow, given the premisses. If the conclusions are certain then the rule is a certain rule. Default rules may interact with certain rules to give results which are not intuitively expected. In *[RC81]* the problem of default rules interacting with universally quantified first order formulae† is examined.

Usually certain rules have the form:

```
(i) All A's are B's
For example, All cows are animals
```

and default rules of the form:

```
(ii) Typically A's are B's
For example, Typically animals do not fly.
```

The default rules and certain rules can interact in the following ways.

```
All A's are B's and Typically B's are C's
       thus Typically A's are C's
For example,   All Eagles are birds
```

† We will refer to universally quantified first order formulae as certain rules

```
          and   Typically birds fly
         thus   Typically Eagles fly

Typically A's are B's and All B's are C's
       thus Typically A's are C's
For example,    Typically American adults own a car
          and   All cars have wheels
         thus   Typically American adults have wheels
```

In both these examples, the conclusions are what is intuitively expected. However, this is not always the case. For example, if given

```
All 21 year olds are adults
Typically adults are married
```

the interaction will give

```
Typically 21 year olds are married.
```

This is not what we would expect since it seems more likely that a 21 year old person would not be married. Thus the interaction gives a conclusion which is not intuitively expected.

DL does not provide a solution to the problem of intuitively wrong conclusions which result from the interaction between default and certain rules.

3 Conflicting Defaults

Another problem with DL is that default conclusions may contradict each other. Defaults can be said to be in conflict if their premisses share a common instance and the conclusion of one contradicts the conclusion of the other.

For example if we are given :

```
Quakers are pacifists (1)
Republicans are not pacifists (2)
John is a Quaker
John is a Republican.
```

John is both a Quaker and a Republican and hence defaults (1) and (2) above, share a common instance. They conflict because selecting the first default will lead to the conclusion John is a pacifist. However, selecting the second default will lead to the conclusion that John is not a pacifist.

The answer to the question of John's warlike nature will depend on our choice of defaults. This is one of the problems with DL since the choice of defaults in arriving at the answer should not affect the answer. DL does not provide a mechanism for dealing with the problem of which default we should choose. This problem with conflicting defaults is significant, especially if a uniform method of arriving at answers is to be maintained.

The problem with conflicting defaults is to decide which of the two possible answers is the better one and if neither is better then what should our answer be. The better answer will be based on the default in which we have a greater degree of confidence.

The main focus of the thesis is to provide a feasible solution to this problem of conflicting defaults.

4 Solutions to the problem of interacting and conflicting defaults

Several researchers *[RC81, Bre91]* have proposed solutions to some of the problems of DL.

4.1 The "Patch" Solution

In *[RC81]* the problem of conflicting and interacting defaults is examined. They identify different cases which, if the default representation is used, lead to counter-intuitive results. They then give alternative default representations ("patches") for each case which would block the erroneous conclusion.

For example we can have the following conflicting defaults

$$\frac{\text{republican}(x)\ :\ \neg\text{pacifist}(x)}{\neg pacifist(x)},$$

$$\frac{\text{quaker}(x)\ :\ \text{pacifist}(x)}{pacifist(x)} \tag{6}$$

which states that typically republicans are not pacifists and typically quakers are pacifists. If we wish no conclusion to be made if John is both a Republican and a Quaker then we replace (6) with the following non-normal defaults:

$$\frac{\text{republican}(x)\ :\ \neg\text{quaker}(x) \wedge \neg\text{pacifist}(x)}{\neg\text{pacifist}(x)},$$

$$\frac{\text{quaker}(x)\ :\ \neg\text{republican}(x) \wedge \text{pacifist}(x)}{\text{pacifist}(x)} \tag{7}$$

This states that typically Republicans who are not Quakers are not pacifists and typically Quakers who are not Republicans are pacifists.

If we know for certain that Republicans are not Quakers and Quakers are not Republicans then we can replace the default representation (7) with the following normal defaults

$$\frac{\text{republican}(x)\ :\ \neg\text{quaker}(x)}{\neg quaker(x)},$$

$$\frac{\text{quaker}(x)\ :\ \neg\text{republican}(x)}{\neg republican(x)} \tag{8}$$

$$\frac{\text{republican}(x) \wedge \neg\text{quaker}(x)\ :\ \neg\text{pacifist}(x)}{\neg\text{pacifist}(x)},$$

$$\frac{\text{quaker}(x) \wedge \neg\text{republican}(x)\ :\ \text{pacifist}(x)}{\text{pacifist}(x)} \tag{9}$$

which states typically Republicans are not Quakers, typically Quakers are not Republicans, typically Republicans who are not Quakers are not pacifists and typically Quakers who are not Republicans are pacifists.

For each case they *[RC81]* devise alternative default representations (**"patches"**).

Despite the "patches" if John is both a Quaker and a Republican we still cannot say anything about his warlike nature. The general pattern of defaults which conforms to our example is

$$\frac{A(x)\ :\ \neg C(x)}{\neg C(x)}\ ,\ \frac{B(x)\ :\ C(x)}{C(x)} \tag{10}$$

Other pairs of defaults fit this pattern. For example:

```
Typically full time students are not employed
Typically adults are employed
John is an adult
John is a full time student.
```

However, in this example, the intuitively correct conclusion would be John is not employed rather than making no conclusion as was the case with the Quaker/Republican example.

They *[RC81]* give another "**patch**" for the adult/full time student example. The problem with the "**patch**" approach is that even if a pair of defaults fits a given pattern, we still cannot be sure which of the alternative default representations ("**patches**") we should use. The choice in the Quaker/Republican example is different from the choice in the adult/full time student example. Thus, it seems necessary for prior information on the nature of the interacting defaults be known before the appropriate default representation can be used. The problem will be more complicated if the addition of new information changes the nature of the interacting defaults.

4.2 The "Named Default" Solution

Brewka *[Bre91]* proposes a slightly different solution to the problem of conflicting default rules. This approach is more elegant than Reiter's since the latter is more specific to the conflicting defaults and requires the explicit representation of the exceptions in the default. In contrast, Brewka's approach seems more general and requires no change to the default.

We illustrate the approach with the following example. Given

```
Typically students are not married  (1)
Typically adults are married        (2)
John is a student
John is an adult
```

we note that the two defaults are in conflict.

We can give preference to (1) in the adult/student example by making it a weak exception of (2). One approach *[RC81]* to achieve this would be to replace (2) by

```
Typically adults who are not students are married
```

However, this is a non-normal default and the approach is impractical since it requires that all exceptions be explicitly mentioned in the default.

Brewka's approach is to give each default a unique name and reason about the applicability of that default. Brewka's approach to this example is to name default rule (2) as R1 and replace default rule (2) with

$$\frac{\text{appl}(R1) \ \wedge \ \text{adult}(x) \ : \ \text{married}(x)}{married(x)} \tag{11}$$

This reads as "***Typically adults for whom default R1 is applicable are married***". We add the "***meta default***"

$$\frac{adult(x)\ :\ appl(R1)}{appl(R1)} \tag{12}$$

This reads as "***Typically default R1 applies to adults***". We then include the blocking formula

$$student(x) \Rightarrow \neg appl(R1) \tag{13}$$

which says "***Default R1 is not applicable to students***".

The advantages of Brewka's approach are that the defaults are normal and new exceptions to defaults will not necessitate changes in the default. The only change necessary would be the addition of a blocking formula for each exception to the default.

One problem with Brewka's approach, which he points out, is that it will not work in the case where the weak exception is derived via a default. In this case a semi-normal representation of *[RC81]* must be used.

Any approach which aims to prioritize defaults suffers from inflexibility since the priority is cast in stone. Ideally we would like to prefer one default over the other depending on the circumstances. In the adult/student example, if we know that John is a mature student returning to school to upgrade his skills then we would probably like to conclude that he is married. However, if John is just eighteen and in his first term in university we would probably prefer to conclude that he is not married.

5 Only True/False Conclusions

DL fails to distinguish between conclusions which are known to be certain and those conclusions which are not certain *[Nut83]*.

For example, if we know

```
Birds fly
Tweety is a bird
Chiree flies
```

then we can conclude

```
Tweety flies
Chiree flies.
```

However we are certain that Chiree flies but there is a possibility that Tweety does not fly. We would reason that Tweety is a bird and typically birds fly hence Tweety flies. The use of the default "birds fly" introduces a degree of doubt in our conclusion.

A more reliable conclusion about Tweety would be

```
There is reason to believe Tweety flies.
```

However, such distinctions are not possible in DL.

Nutter *[Nut83]* argues that if a distinction between guarded statements of the form **"there is reason to suppose P"** and **"P"** are made then a monotonic logic can deal with reasoning from default generalizations. However, Nutter does not give an example to illustrate this point, nor elaborates on it.

6 Conclusion

In this chapter we have shown that there are many problems with DL. Default rules can interact to give intuitively wrong conclusions. Common instances of default rules, as well as certain and default rules, can contradict each other.

We have shown some proposed solutions *[RC81, Bre91]*. We also discuss the problems associated with the lack of distinction between "***There is reason to believe P***" and "***P***" in DL.

In the next chapter, we present our approach to commonsense reasoning.

Chapter 4: SOLUTION OUTLINE

In this chapter we outline our approach to computer based common sense reasoning. We give details of our solution with examples and describe its advantages.

1 Introduction

In human common sense reasoning we arrive at conclusions based on the available information. For example, we know most birds fly and if Tweety is a bird then we conclude Tweety flies. However, we are not surprised if later we find Tweety does not fly since we are aware that our conclusion is not certain. We know this since the information we use, "***Most birds fly***", is not certain. On the other hand, we would be very surprised if we found out that Daisy the cow flies since we are certain cows don't fly. The degree of certainty we have in our conclusion is dependent on the degree of certainty we have in the information used to arrive at the conclusion.

It is clear that we can attach some degree of certainty to each unit of information we have. However, it is not clear what is the nature of this value, nor how we arrive at its value, nor how we combine these values.

It is clear that we need some measure of certainty in computer based common sense reasoning systems. Therefore we propose to have an explicit certainty factor attached to each statement in our system.

2 Our Solution

2.1 Add Certainty Factor

In our solution, we use a rule based approach and attach a certainty factor (CF) to each rule and fact in our knowledge base. The CF allows for the following different types of information:

- **Certain Rules**: rules where the conclusion is indisputably true.
  For example, all birds are mammals.
- **Blocking**[‡] **Rules**: rules where the conclusion is indisputably false.
  For example, penguins don't fly.
- **Default Rules**: rules where there is some doubt about the certainty of the conclusion.
  For example, most birds fly or typically Canadians play hockey.
- **Certain Facts**: facts known to be indisputably true.
  For example, the earth is round.
- **Uncertain Facts**: facts where there is some doubt about their truth or falsehood.
  For example, cholesterol causes heart disease or there is a god.
- **Blocking Facts**: facts known to be indisputably false.
  For example, the earth is flat.

A certainty factor value represents a degree of certainty in the statement to which it is attached. In our system, we use a scale from zero to plus or minus one to represent our degree of certainty in a clause.

We are certain that a clause is true[§] if has a CF of plus one or minus one.

[‡] The choice of the name "Blocking" will be made clear when we explain our resolution strategy

[§] The semantics for the clauses with a CF are given in chapter 5.

For example,

$$\text{bird(tweety)} \leftarrow : +1.0$$

means we are certain Tweety is a bird.

$$\text{bird(john)} \leftarrow : -1.0$$

means we are certain John is not a bird.We are certain that a clause is false if it has a zero certainty factor. For example,

$$\text{flies(X)} \leftarrow \text{cow(X)} : 0$$

which means we have no certainty in the statement that cows fly.

The degree of certainty in the truth of a clause increases as we approach the limits ($\pm$ 1). The degree of certainty decreases as we approach zero and a CF of zero means no certainty in the truth of the statement, i.e. the statement is false.

We use a calculus to update the CF as we reason about conclusions. Resolution refutation *[Rob65]* is used to arrive at a conclusion. The syntax, semantics and proof theory for our solution are explained in the next chapter. We now give some examples which illustrate our solution.

2.2 Examples: Representation

In order to make our examples clearer, we give representations for some rules and facts. The certain fact, ***Tweety is a bird***, is represented as:

$$\text{bird(tweety)} \leftarrow : 1.0$$

The blocking fact, ***Jasper is not a bird***, is represented as:

bird(jasper) ← : - 1.0

This means that we are certain Jasper is not a bird.

The uncertain fact, ***Zak is most likely a bird***, is represented as:

bird(zak) ← : 0.8

However the actual value of the CF will depend on what is meant by **most likely**. The uncertain fact, "***It is unlikely that Jim is a bird***", is represented as:

bird(jim) ← : - 0.6

The actual value of the CF will depend on what is meant by **unlikely**. The certain rule, "***Birds are mammals***", is represented as:

```
mammal(X) <- bird(X) : 1.0
```

and reads if X is a bird then we can conclude with absolute certainty that X is a mammal. The blocking rule, "***Dead things don't fly***", is represented as:

```
   flies(X) <- dead(X) : - 1.0
```

and reads *if X is dead then we can not conclude that X flies.*

The default rule, "***Most birds fly***", is represented as:

```
   flies(X) <- bird(X) : 0.9
```

and reads *if X is a bird then we can conclude with a 0.9 degree of certainty that X flies.* Here we represent the information that most, but not all, birds fly. The CF varies according to what we mean by "**most**".

It is important to realize that the certainty increases as we approach a limit. For example, given

$\text{bird(bill)} \leftarrow : -0.9$

$\text{bird(ben)} \leftarrow : 0.8$

then we are more certain that Bill is not a bird than Ben is a bird. This idea is an important part of our resolution strategy.

2.3 Examples : Benchmark

In *[Lif89]*, some benchmark problems for non-monotonic reasoning systems are presented. In this section we illustrate how our system handles some of these problems. The assumptions are those given in the problem. The benchmark conclusions are what the system should be able to derive. The examples have been modified to the extent that we have added certainty factors to the assumptions for illustrative purposes.

2.3.1 Basic Default Reasoning

The assumptions are:

```
Blocks A and B are heavy
Heavy blocks are normally located on the table
A is not on the table
```

The benchmark conclusion is:

```
B is on the table.
```

In our system, this is represented as

heavy(block-A) $\leftarrow$: 1.0

heavy(block-B) $\leftarrow$: 1.0

on-table(X) $\leftarrow$ heavy(X) : 0.75

ontable(block-A) $\leftarrow$: - 1.0

If we query the system: *What is on the table ?* The conclusion we get is:

block-B $\leftarrow$: 0.75

which means that we can conclude that block B is on the table with 0.75 certainty. Note that the benchmark conclusion is not as realistic as our conclusion since the latter states it is likely that block B is on the table and the former states block B is on the table. Our system distinguishes between certain answers and likely answers.

2.3.2 Reasoning with Several Defaults

The assumptions are:

```
Blocks A and B are heavy
Heavy blocks are normally located on the table
Heavy blocks are normally red
A is not on the table
B is not red
```

The benchmark conclusions are:

```
B is on the table
A is red
```

In our system this is represented as

heavy(block-A) $\leftarrow$: 1.0

heavy(block-B) $\leftarrow$: 1.0

on-table(X) $\leftarrow$ heavy(X) : 0.75

red(X) $\leftarrow$ heavy(X) : 0.8

ontable(block-A) $\leftarrow$: - 1.0

red(block-B) $\leftarrow$: - 1.0

If we query the system: *What is on the table ?* The conclusion we get is:

block-B ← : 0.75

If we query the system: *What is red ?* The conclusion we get is:

block-A ← : 0.8

which means we can conclude with 0.8 certainty that block A is red. The answers generated by our system are more realistic since they indicate that the conclusions are likely but not certain.

2.3.3 Priorities between defaults

The assumptions are:

```
Jack asserts that block A is on the table
Mary asserts that block A is not on the table
When Jack asserts something he is normally right
When Mary asserts something she is normally right
Mary's evidence is more reliable than Jack's
```

The benchmark conclusion is:

```
Block A is not on the table
```

In our system this is represented as:

ontable(X) ← asserts(mary, not-on-table-A) : - 1.0

ontable(X) ← asserts(jack, on-table-A) ← : 1.0

correct(X) ← asserts(jack,X) : 0.75

correct(X) ← asserts(mary,X) : 0.9

The fact that Mary's evidence is more reliable is very naturally represented in the certainty factors. It is a very efficient way to represent priorities between defaults. In *[Lif89]* two approaches to this bechmark example is given, one based on prioritized circumscription. However, in the paper one approach is said to be "*not quite 'declarative'*" and the other noted as noted "as not being sufficient". These examples just emphasize the difficulty in representing priorities between defaults.

Our system is particularly well suited to this type of problem. If we query the system with: *What can we correctly assert ?* The conclusion we get is:

```
on-table-A : - 0.9
```

which means it is very unlikely that block A is on the table. This is since Mary's evidence is more reliable than Jack's.

2.4 Example: Conflicting Defaults

In the general case we have

```
Typically A's are C's
Typically B's are not C's.
```

These defaults are in conflict if we have a joint instance of A and B. For example, the defaults:

```
Typically adults are married         (1)
Typically students are not married   (2)
```

are in conflict since we can have a joint instance:

```
John is an adult
John is a student
```

In human common sense reasoning, when defaults are in conflict we tend to prefer the one in which we have a greater degree of certainty. Suppose we are certain that it is more likely that students are **not** married than adults are married We can represent this as:

married(X) ← adult(X) : 0.75

married(X) ← student(X) : - 0.85

adult(john) ← : 1.0

student(john) ← : 1.0

If we query the system with: *Who is married ?* The conclusion we get is:

```
John : - 0.85
```

which means it is very unlikely that John is married.

3 Conclusion

In this chapter we have outlined our solution to some NMR problems. We have shown with examples how certainty factors can be used to solve some benchmark problems. In the next chapter we present the syntax, semantics and proof theory for our solution.

Chapter 5: THE CFDL LOGIC

1 Introduction

One of the problems with Default Logic is that it does not distinguish between certain and uncertain conclusions. This is directly related to the lack of distinction between default rules and certain rules.

In our approach, we make the distinction clear by attaching certainty factors to the rules. This idea is generalized to have a CF attached to every unit of information in the Knowledge Base.

The precise meaning of the new representation is defined in this chapter. Care is taken so that the new logic will degenerate to Default Logic. We also formulate a proof theory, based on resolution refutation *[Rob65]*, which takes advantage of our new syntax to provide the most certain answer to any query.

We define the syntax, semantics and proof theory for our system. We call this new logic "***certainty factor default logic***" (CFDL) since it associates a degree of certainty with each piece of information in our knowledge base. CFDL degenerates to DL at the limits (where the certainty factors are either plus or minus one).

2 Syntax

We have the following:

a set of constants $C = \{c_1, c_2, ... , c_n\}$;

a set of variables $V = \{v_1, v_2, ... ,v_n\}$;and

a set of n-ary ($n \geq 1$) predicates $P = \{p_1, p_2, ... , p_n\}$.

A **term** is defined to be either a constant or a variable.

An **atom** is defined as

$$p(t_1, t_2, \ldots\ldots, t_n)$$

$n \geq 1$, where $p \in P$ and each t_i is a term.

A **clause** is defined as

$$q \leftarrow a_1, a_2, \ldots\ldots, a_n : cf$$

where $n \geq 0$, $cf \in \Re$, $-1 \leq cf \leq +1$, and $q, a_1, a_2, \ldots\ldots, a_n$ are atoms.

Example : the following is a clause

$$flies(x) \leftarrow bird(x), winged(x) : 0.95$$

Informally, this means that we are reasonably certain that if x is a bird and has wings then it flies. Similarly,

$$flies(x) \leftarrow dead(x) : -0.99$$

is intended to mean that dead things do not fly.

Frequently we need to express rules such as

$$\text{if q then not p} \qquad (14)$$

which, in clausal form is written

$$\leftarrow p, q \qquad (15)$$

In our representation we use the equality

$$p \leftarrow q : -cf \equiv \leftarrow p, q : cf \qquad (15)$$

as a syntactic mechanism for indicating that the rule is "*about p*".

For example, the clause

$\leftarrow$ male(x), female(x) : 1

means that nothing can be both male and female. However,

male(x) $\leftarrow$ female(x) : -1

says that something is not male if it is female. This is useful in certain situations as we shall see later.

3 Semantics

An interpretation I = (W, D) for a knowledge base KB consists of W, an non-empty set of objects and a denotation function D such that for every constant c, D(c) is an object in W and for every predicate p, D(p) is an n-ary relation in W.

A valuation is a function such that for every constant c in KB, V(c) = D(c); for every predicate p in KB, V(p) = D(p); and for every variable v, V(v) is an object in W.

Given an interpretation I = (W, D) where D is defined for all constants and predicates in KB and a valuation V then V **satisfies the atom**

$p(t_1, t_2, \ldots\ldots, t_n)$ if $<v(t_1), v(t_2), \ldots, v(t_n)> \in V(p)$ in W.

Example: the knowledge base contains the following clauses flies(bill) $\leftarrow$: 1

flies(ben) $\leftarrow$: 1

flies(tweety) $\leftarrow$: -1

W comprises the objects Bill, Ben and Tweety. D(bill) = Bill, D(ben) = Ben and D(tweety) = Tweety. D(flies) = {Bill, Ben}. V(flies) = {Bill, Ben}. The atom **flies(bill)** is satisfied since V(bill) = D(bill) = Bill and Bill ϵ V(flies).

Given an interpretation I, a valuation V and the clause

$$C = q \leftarrow a_1, a_2, \ldots, a_n : cf,$$

where $n \geq 0$, $0 \leq cf \leq 1$ and $q, a_1, a_2, \ldots, a_n$ are atoms then **V satisfies C with certainty cf** if

(a) V satisfies q or

(b) V fails to satisfy at least one a_i or both

Otherwise V does not satisfy C.

Given an interpretation I, a valuation V and the clause

$$C = q \leftarrow a_1, a_2, \ldots, a_n : cf,$$

where $n \geq 0$, $-1 \leq cf < 0$ and $q, a_1, a_2, \ldots, a_n$are atoms then **V satisfies C with certainty |cf|** if

(a) V fails to satisfy q or

(b) V fails to satisfy at least one a_i or both

Otherwise V does not satisfy C.

A clause C is **true with certainty** |cf| if every valuation satisfies C. Otherwise it is false.

Example: We have a knowledge base with the following information

canadian(bill) ← : 1

canadian(ben) ← : 1

plays-hockey(bill) ← : 1

plays-hockey(ben) ← : 1

plays-cricket(bill) ← : 1

plays-cricket(ben) ← : 1

The clause

canadian(x) ← plays-hockey(x) : 1

is true since every valuation (x = bill, ben) satisfies it. However, the clause

canadian(x) ← plays-cricket(x) : −1

is false since at least one valuation (x = ben) fails to satisfy it.

4 Proof Theory

The resolution rule comprises two mutually exclusive cases:

Case 1: The certainty factors of the two input clauses have the same sign, both positive or negative. In this case, given

$r \leftarrow p, a_1, a_2, \ldots\ldots, a_m : cf_1$ and

$p \leftarrow b_1, b_2, \ldots\ldots, b_n : cf_2$

where $m \geq 0$ and $n \geq 0$ then we can deduce

$$r \leftarrow a_1, a_2, \ldots\ldots, a_m, b_1, b_2, \ldots\ldots, b_n : cf_3$$

where $cf_3 = cf_1 \otimes cf_2$ and $\otimes$ is defined by the calculus being used which ensures that $|cf_3| \leq \min(|cf_1|, |cf_2|)$. (That is the resolvant can never be more certain than either of the input clauses.)

Case 2: The certainty factors of the two input clauses have opposite signs, one is positive the other negative. In this case, given

$q \leftarrow a_1, a_2, \ldots\ldots, a_m : cf_1$ and

$q \leftarrow b_1, b_2, \ldots\ldots, b_n : cf_2$

where $m \geq 0$ and $n \geq 0$ then we can deduce

$$\leftarrow a_1, a_2, \ldots\ldots, a_m, b_1, b_2, \ldots\ldots, b_n : cf_3$$

where $cf_3 = cf_1 \otimes cf_2$ and $\otimes$ is defined by the calculus being used which ensures $|cf_3| \leq \min(|cf_1|, |cf_2|)$. Note that cf_3 will be positive to concur with the usual resolution rule.

Example of Case 1: The knowledge base contains the following clauses (numbered for reference):

flies(x) ← bird(x), winged(x) : 1 (1)

bird(tweety) $\leftarrow$: 1 (2)

winged(tweety) $\leftarrow$: 1 (3)

The query asks "*does Tweety fly?*" and is expressed as the clause

$\leftarrow$ flies(tweety) : 1 (4)

Resolution* proceeds as follows:

Clauses Resolved	Resolvent
(4) & (1)	$\leftarrow$ bird(tweety), winged(tweety) : 1 (5)
(5) & (2)	$\leftarrow$ winged(tweety) : 1 (6)
(6) & (3)	{} $\leftarrow$ {} : 1

The answer is "**tweety flies**".

As another example of Case 1 consider the following clauses:

flies(x) $\leftarrow$ dead(x) : -1 (1)

dead(bill) $\leftarrow$: 1 (2)

In this case we can prove that Bill does not fly by expressing the query as

$\leftarrow$ flies(bill) : -1 (3)

and resolution proceeds as follows:

(3) & (1)	$\leftarrow$ dead(bill) : 1 (4)
(4) & (2)	{} $\leftarrow$ {} : 1

* A simple calculus, which multiplies the two input cf numbers, is shown for illustrative purposes.

In this case the answer is "Bill does not fly".

Example of Case 2: Consider the knowledge base

hot(x) $\leftarrow$ red(x) : 1 (1)

hot(coals) $\leftarrow$: -1 (2)

We can prove that the coals are not red by using the query

red(coals) $\leftarrow$: 1 (3)

Resolution proceeds as

(3) & (1)	hot(coals) $\leftarrow$: 1 (4)
(4) & (2)	{} $\leftarrow$ {} : 1

The answer is "**the coals are not red**".

Note that because of the way in which the resolution rule is defined we can use the following equivalence for queries:

$$p \leftarrow : 1 \equiv \leftarrow p : -1 \tag{16}$$

The resolution rule allows for a modified Modus Ponens and Modus Tollens. Modus Ponens now becomes:

$$\begin{array}{l} p \leftarrow \ : cf_1 \\ q \leftarrow p \ : cf_2 \\ \hline q \leftarrow \ : cf_3 \end{array} \tag{16}$$

where $|cf_3| \leq \min(|cf_1|, |cf_2|)$. When $cf_1 = cf_2 = 1$ then we recover the usual Modus Ponens (and $cf_3 = 1$).

Modus Tollens becomes:

$$\begin{array}{l} q \leftarrow p \ : \ cf_1 \\ q \leftarrow \ : \ -cf_2 \\ \hline \leftarrow p \ : \ cf_3 \end{array} \tag{16}$$

where $|cf_3| \leq \min(|cf_1|, |cf_2|)$. Modus Tollens is recovered when $cf_1 = cf_2 = 1$.

Queries are processed using resolution refutation. The resolution strategy ensures that the most certain answer is (usually)[†] derived without evaluating all possible answers. The strategy chooses the two input clauses which will produce the most certain resolvent. An assumption is that the resolvent can never be more certain than the most certain input clause. The strategy is outlined below:

Step 1 The current goal G is the (sub) query

$$\leftarrow R_1.R_2.\dots\dots.R_n \ : \ cf \tag{17}$$

where $n \geq 1$, $cf = \pm 1$.

Step 2 Find the most certain clause C in the knowledge base which will unify with the leftmost atom of G.

Step 3 Resolve G and C to produce the resolvent R. In constructing R the remaining antecedents of the most certain clause (G or C) are placed to the left of the remaining antecedents of the other clause.

If C does not exist then backtrack.

[†] In the worst case all possible answers will be examined but we expect that this will not happen frequently.

Step 4 If R is the empty clause and the most certain answer then stop and report answer.

Step 5 If R is not the empty clause then find G and C in the knowledge base such that they produce the most certain resolvent possible. G must be a previous goal or resolvent.
Repeat procedure from step 3.

Backtrack Go to step 5
If it is not possible to backtrack then report failure to user.

Example: we have the following clauses (numbered for illustration):

flies(x) $\leftarrow$ bird(x) : 0.8 (1)

flies(x) $\leftarrow$ bat(x) : 0.9[illegible] (2)

flies(x) $\leftarrow$ dead(x) : -0.99 (3)

bird(ben) $\leftarrow$: 0.7 (4)

bat(ben) $\leftarrow$: 0.9 (5)

bird(tweety) $\leftarrow$: 1 (6)

dead(tweety) $\leftarrow$: 1 (7)

The query "*does Tweety fly*" is expressed as the clause

$$\leftarrow \; flies(tweety) \; : \; -1 \qquad (0)$$

since resolving with (3) will give the most certain resolvent. Resolution proceeds as follows:

Input clauses	Resolvent
(0) & (3)	← dead(tweety): 0.99 (8)
(8) & (7)	[] : 0.99

No other refutation will lead to a more certain answer. Therefore we report **"Tweety does not fly"**.

To handle the query "*who flies ?*" we need the two goal clauses

$$\begin{array}{ll} \leftarrow flies(x) : -1 & (0) \\ \leftarrow flies(x) : 1 & (0') \end{array} \qquad (19)$$

Resolution proceeds as follows:

Input clauses	Resolvent	action
(0) & (3)	← dead(tweety): 0.99 (8)	
(8) & (7)	[] : 0.99 x=tweety	Record "tweety does not fly: 0.99" & backtract
(0′) & (2)	← bat(x) : 0.9 (9)	store clause as choice point
(0′) & (1)	← bird(x) : 0.8 (10)	
(10) & (6)	[] : 0.8 x=tweety	answer discarded since more certain answer available
(10) & (4)	[] : 0.72 x=ben	report "Ben flies : 0.72"
(9) & (5)	[] : 0.63 x=ben	more certain previously for Ben, so this answer is discarded. Terminate since backtracking not possible

5 Properties of Clauses

Given a set of clauses C then for each clause c_i we have a measure of the certainty, denoted $CF(c_i)$, that c_i is true. $CF(c_i)$ takes its values from the real interval $[-1, +1]$. It is not a degree of truth. Each clause is either true or false but given the information available, which may be incomplete, it is an estimate of the certainty that the clause is true. Thus we have the following axioms:

$$|CF(c_i)| = 1 \text{ if } c_i \text{ is logically true} \tag{20}$$

$$|CF(c_i)| = 0 \text{ if } c_i \text{ is logically false} \tag{21}$$

For example, the statement $p \vee \neg p$ is logically true. It is represented as $p \leftarrow p$ in clausal form so $|\text{CF}(p \leftarrow p)| = 1$. The empty clause is logically false so $|\text{CF}(\{\} \leftarrow \{\})| = 0$.

For all clauses p and q we have

$$|\, CF(p \wedge q) \,| \leq min(|\, CF(p) \,|, |\, CF(q) \,|) \tag{22}$$

$$|\, CF(p \vee q) \,| \leq max(|\, CF(p) \,|, |\, CF(q) \,|) \tag{23}$$

Using the equality

$$p \leftarrow q : -1 \equiv \leftarrow p, q : 1 \tag{23}$$

our resolution rule can be expressed as

$$\begin{array}{l} CF(p \vee q) = \alpha \\ CF(\neg p \vee r) = \beta \\ \hline CF(q \vee r) \leq min(\alpha, \beta) \end{array} \tag{23}$$

We do not insist on the equality

$$CF(\neg p) = 1 - CF(p) \tag{24}$$

Certain inconsistencies are allowed and handled by using the most certain clause of any conflicting clauses. For example, from

$\text{flies}(x) \leftarrow \text{bird}(x) : 0.95$

$\text{flies}(x) \leftarrow \text{dead}(x) : -1$

$\text{bird}(\text{tweety}) \leftarrow : 1$

$\text{dead}(\text{tweety}) \leftarrow : 1$

we can deduce both

$\text{flies}(\text{tweety}) \leftarrow : 0.95$

$\text{flies}(\text{tweety}) \leftarrow : -1$

However, we use the latter fact, tweety does not fly, as it is more certain. Essentially we work with a consistent subset of the knowledge base.

6 Conclusion

In this chapter we have given details of our CFDL. In the next chapter we give some of the implementation details of our system.

Chapter 6: THE RESOLUTION STRATEGY

1 Introduction

To test our hypothesis and to have a testbed for different calculi of uncertainty the CFDL system has been implemented in Quintus Prolog *[Qui89a]* with ProWindows *[Qui89b]*.

The knowledge base consists of clauses of the form

$$q \leftarrow p_1, p_2, \ldots\ldots, p_n : cf \tag{25}$$

where $n \geq 0$, $-1 \leq cf \leq 1$ and q, p_i are atoms. Queries are of the form

$$\leftarrow r_1, r_2, \ldots\ldots, r_n : cf \tag{26}$$

where $n \geq 1$ and cf is either -1 or $+1$.

The proof theory is based on resolution refutation. The resolution strategy ensures that the most certain answer is computed and displayed to the user. This can usually be done without evaluating all answers. The strategy always resolves the two clauses which will give the most certain resolvent, thereby ensuring that the most certain path in the resolution tree is always followed. A basic assumption here is that the resolvent can never be more certain than either of the input clauses. Any calculus which adheres to this assumption can be used with the strategy.

2 The resolution strategy algorithm

The following lists are used to hold information essential to the working of the algorithm:

Choice-points-list	This list stores all the choice points for every goal. It is an efficient means of remembering where to backtrack to, if

necessary. In this algorithm, it is necessary to backtrack to the next most certain clause, not to the next in sequence in the knowledge base as is the case with Prolog for example. The list is ordered by decreasing absolute certainty factor values. The general format for each element is $[G, C_1, C_2, \dots, C_n]$ where G is the goal clause and each C is a clause that will unify with G.

Active-list The active-list provides a one-step-look-ahead guide to the "best" goal clause at any time. It is ordered so that the top entry is the goal clause which when resolved will give the most certain resolvent possible. It is this list, together with the choice points list, which enables the algorithm to produce the most certain answer without evaluating all possible answers. The general format for each entry is $[G, CF_{G \otimes C}]$ where G is a previous goal or resolvent and $G \otimes C$ is the certainty factor of G and C resolved. The list is ordered by decreasing absolute values of $G \otimes C$.

Answer-list Because the knowledge base may contain inconsistent and incomplete data then it is possible that we may get more than one "answer" for a particular binding. For example, in response to "who flies" the algorithm may produce both "Tweety : 0.9" and "Tweety : 0.75" as answers. We want only to report the most certain answer to the user. To do this a list is maintained, containing the most certain answer for each binding. Whenever an answer is produced the list is checked to see if a more certain answer has already been reported to the user. If so then the new answer is ignored.

All-answers-list Every answer produced by the algorithm (for a particular query) is placed in this list. This list is used mainly for tracing and debugging purposes.

The resolution strategy is implemented as follows:

Step 1 The current goal, G is the query

$$\leftarrow r_1, r_2, \ldots\ldots, r_n : cf \qquad (27)$$

where $n \geq 1$ and each r_i is an atom.

Step 2 First, check to see if G is in the choice-points list. If it is then pop the element [G, C_1, C_2, ... , C_n]. Go to step 3.

Otherwise, find the most certain clauses in the knowledge base which will unify with the leftmost atom of G. To do this the knowledge base is searched for all such clauses. If there is no such clause then BACKTRACK. Otherwise order the clauses by decreasing absolute certainty factor values to get the list [C_1, C_2, ... , C_n].

Step 3 G is resolved with C_1 to produce the most certain resolvent, R. In constructing R the remaining antecedents of the most certain clause (G or C_1) are placed to the left of the remaining antecedents of the other clause.

Next [G, C_2, ... , C_n] is placed on the choice-points list.

Next [G, $CF_G \otimes CF_{C2}$], ... , [G, $CF_G \otimes CF_{Cn}$] is placed on the active-list.

Step 4 If R is the empty clause then REPORT. Place binding(s) and certainty factor on the answer-list, if necessary. Place binding(s) and certainty factor on the all-answers-list. The user may force backtracking at this point to get more answers.

If R is not the empty clause then place [R, CF_R] on the active-list and continue to step 5.

Step 5 Find G and C such that the resolvent will be the most certain possible. G is in the first element of the active-list.

If G is found then repeat the procedure from step 2, otherwise report "failure" to the user.

BACKTRACK Repeat the procedure from step 5.

REPORT Report answer to user, checking with answer-list first.

3 An example

This example illustrates some of the features of the algorithm. Clauses are numbered for illustration. The knowledge base contains the following clauses:

flies(x) ← bird(x) : 0.8 (1)

flies(x) ← dead(x) : − 0.99 (2)

bird(tweety) ← : 1 (3)

dead(tweety) ← : 1 (4)

bird(ben) ← : 0.8 (5)

bird(bill) ← : 0.9 (6)

The query is "who flies ?". The query is evaluated as follows:

Step in Algorithm	Action
1	G is the clause " ← flies(x) : cf " where cf has not yet been determined.
2	G is not in the choice-points list. Search KB to produce list of clauses [(2), (1)].
3	G is the clause " ← flies(x):−1" . C is the clause (2). R is the clause "← dead(x):0.99" . [← flies(x):1, flies(x) ← bird(x) : 0.8] is placed on the choice-points-list. [← flies(x):1, 0.8] is placed on the active- list.
4	[← dead(x): 0.99, 0.99] is placed on the active list.
5	G is "← dead(x):0.99" .
2	G is not in the choice-points list. Search KB to find clause (4). C is "dead(tweety) ← : 1".

Step in Algorithm	**Action**
3	"← dead(x):0.99" and "dead(tweety) ← : 1" are resolved to produce the empty clause, the binding x=tweety and cf = 0.99.
4	R is empty clause. [tweety, −0.99] is placed on answer-list and all-answers-list, since we have proved that tweety does NOT fly. Nothing is reported to the user. The system automatically backtracks.
5	G is the first element in the active-list, " ← flies(x):1".
2	G is in the choice-points-list. C is "flies(x) ← bird(x) : 0.8".
3	" ← flies(x):1" and "flies(x) ← bird(x) : 0.8" are resolved to produce R, "← bird(x) : 0.8".
4	[← bird(x) : 0.8, 0.8] is placed on the active-list
5	G is "← bird(x) : 0.8".
2	G is not in the choice-points list. Search KB to produce list of clauses [(3), (6), (5)].

Step in Algorithm	**Action**
3	"← bird(x) : 0.8" and "bird(tweety) ← : 1" are resolved to produce the empty clause, the binding x=tweety and cf = 0.8. [← bird(x) : 0.8, bird(bill) ← : 0.9, bird(ben) ← : 0.8] is placed on choice-point-list. [← bird(x) : 0.8, 0.72, 0.64] is placed on active-list.
4	In REPORT we find that we already have a more certain answer for the binding tweety. Nothing is reported to the user. [tweety, 0.8] is placed in all-answers-list. System automatically backtracks.
5	G is the clause "← bird(x) : 0.8".
2	G is in choice-points-list. C is "bird(bill) ← : 0.9".
3	"← bird(x) : 0.8" and "bird(bill) ← : 0.9" are resolved to produce the empty clause, the binding x=bill and cf = 0.72. [← bird(x) : 0.8, bird(ben) ← : 0.8] remains on choice-point-list. [← bird(x) : 0.8, 0.64] remains on active-list.
4	We report the answer " bill flies with certainty 0.72". The user forces backtracking to produce another answer.
5	G is the clause "← bird(x) : 0.8".
2	G is in choice-points-list. C is "bird(ben) ← : 0.8".

Step in Algorithm	**Action**
3	"← bird(x) : 0.8" and "bird(ben) ← : 0.8" are resolved to produce the empty clause, the binding x=ben and cf = 0.64.
4	We report the answer " ben flies with certainty 0.64". The user forces backtracking to produce another answer.
5	The active-list is empty. Backtracking fails and the failure to find another answer is reported to the user.

4 Conclusion

In this chapter we have outlined, in some detail, the resolution strategy. In the next chapter we will give the conclusions and future work.

Chapter 7: CONCLUSIONS & FUTURE WORK

In this chapter we discuss some of the findings of our research, outline some future work and show how the conclusions supports the thesis statement.

1 Discussion

The CFDL system has the following features:

1. It gives an efficient way of representing priorities between defaults.
2. It can solve some of the benchmark problems in NMR.
3. The algorithm is "*lazy*" in the sense that all solutions need not be examined to get the most certain answer.
4. The system is a test bed for different calculi.

These points are discussed below as well as our rationale behind the choice of one certainty factor.

1.1 Efficient Representation

In some systems *[Poo92, RC81]* defaults are either labelled or rewritten. The CFDL allows for efficient representation of defaults without the need for tagging the default or rewriting the default. Given:

```
Typically birds fly
Typically emu's don't fly
All emu's are birds
All robin's are birds
```

```
Tweety is a bird
Polly is an emu
Cohen is a robin
```

We can derive:

```
Tweety and Cohen fly
Polly does not fly
```

In Theorist *[Poo92]* the necessary representation will be:

```
default birdsfly(X) : flies(X) <- bird(X).
constraint not birdsfly(X) <- not flies(X).
default emusdontfly(X) : not flies(X) <- emu(X).
constraint not emusdontfly(X) <- flies(X).
constraint not birdsfly(X) <- emu(X).
fact bird(X) <- emu(X).
fact bird(X) <- robin(X).
fact bird(tweety).
fact emu(polly).
fact robin(cohen).
```

The default rule "***Typically birds fly***" is represented by

```
default birdsfly(X) : flies(X) <- bird(X).
```

with the exceptions to this rule being:

emus, represent by the constraint

```
constraint not birdsfly(X) <- emu(X).
```

and things known not to fly, represented by the constraint:

```
constraint not birdsfly(X) <- not flies(X).
```

Similarly, the default "***Typically emus don't fly***" is represented by

```
default emusdontfly(X) : not flies(X) <- emu(X).
```

with the exception things known to fly represented by

```
constraint not emusdontfly(X) <- flies(X).
```

The certain rules "***All emus are birds***" and "***All robins are birds***" are represented by

```
fact bird(X) <- emu(X).
fact bird(X) <- robin(X).
```

The certain facts "***Polly is an emu***" and "***Cohen is a robin***" are represented by

```
fact emu(polly).
fact robin(cohen).
```

This representation generates the expected conclusions but note how we can represent the same information in CFDL.

```
flies(X) <- bird(X) : 0.9
flies(X) <- emu(X) : -0.95
bird(X) <- emu(X) : 1.0
bird(X) <- robin(X) : 1.0
bird(tweety) <- : 1.0
emu(polly) <- : 1.0
robin(cohen) <- : 1.0
```

If we query our system with: *Who flies ?* The conclusions will be:

```
Tweety : 0.9
Polly : 0.9
Cohen : - 0.95
```

which means that ***it is very likely that Tweety and Polly fly and very unlikely that Cohen flies.***

In comparison to Theorist, our representation is more compact. The difference between defaults and facts are encoded in the CF. The need for representing constraint rules is eliminated. The effect of the constraints is accomplished by our algorithm on examining the CF.

Our representation is compact and provides an elegant way of representing priorities between defaults. Also, modification to these priorities can be easily made in contrast to the solutions suggested by *[Bre91, RC81, Lif89]*.

1.2 Solves some Benchmark problems in NMR

The CFDL system can handle the following benchmark examples: basic default reasoning, reasoning with several defaults, priorities between defaults and conflicting defaults (as shown in chapter 4).

Our system also provides a solution to the problem of there being no distinction between certain and default answers. For example, given

```
Typically birds fly
Tweety is a bird
Chiree flies
```

In traditional systems no distinction is made between the conclusions "Tweety flies" and "Chiree flies". However, it is clear that we have more confidence that Chiree flies than Tweety flies. This distinction can be made in our system. Our representation will be:

$\text{flies}(X) \leftarrow \text{bird}(X) : 0.9$

$\text{bird}(\text{tweety}) \leftarrow\ : 1.0$

$\text{flies}(\text{chiree}) \leftarrow\ : 1.0$

Our answers to the query "Who flies ?" will be:

```
Chiree : 1.0
Tweety : 0.9
```

which means we are certain that Chiree flies and it is very likely but not certain that Tweety flies.

In our solution, we can also distinguish between negative answers of the forms: *certainly not*, *unlikely* and *no by closed world assumption (CWA)*. This is possible since negative information can be explicitly represented. For example, if we know Tim does not fly, this can be represented as: "***flies(tim) : – 1.0***". Hence if we are asked if Tim flies we can answer certainly not. If we are asked *if Tam flies* we cannot say *certainly not* since there is no explicit representation for "***flies(tam)***". However, we can use the CWA assumption to give the answer *no* knowing we are not certain of this answer. Our KB can also have information of the type "***flies(tony) : 0.4***". In such a case the answer as to if Tony flies will be *it is unlikely that Tony flies*.

1.3 Lazy Algorithm

Our algorithm is "*lazy*[*]" since it produces the best answer without evaluating all possible answers. This is possible since our criterion for best answer is the answer which is most certain and our calculus does not allow conclusions to be more certain than either of the premises. For example, if we are given:

flies(X) ← bat(X) : 0.95 (1)

flies(X) ← bird(X) : 0.9 (2)

flies(X) ← parrot(X) :0.7 (3)

bird(X) ← robin(X) : 1.0 (4)

bird(X) ← parrot(X) : 1.0 (5)

robin(polly) ← : 1.0 (6)

bat(tim) ← : 1.0 (7)

[*] We do not mean 'lazy' in the functional programming sense.

parrot(pat) ← : 0.5 (8)

Our most certain answer as to ***who flies*** is Tim. Our system will generate the answer without looking at (4) and (5) since we know from (1) that any answer along that solution path can not have a CF higher than 0.9. We do not need to examine (5) and (8) either since from (3) any solution along that path will be smaller than 0.7. Our search tree will be

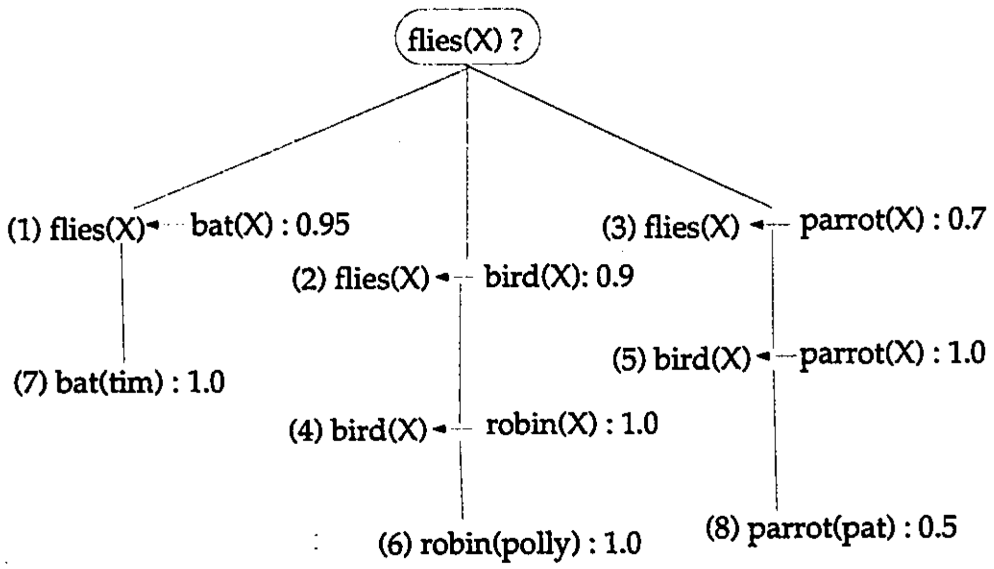

flies(X) ?
(1) flies(X) ← bat(X) : 0.95
(2) flies(X) ← bird(X): 0.9
(3) flies(X) ← parrot(X) : 0.7
(7) bat(tim) : 1.0
(5) bird(X) ← parrot(X) : 1.0
(4) bird(X) ← robin(X) : 1.0
(6) robin(polly) : 1.0
(8) parrot(pat) : 0.5

Figure 1.1 *Search tree for Who Flies*

1.4 Testbed for Calculi

For test purposes, a very simple calculus of uncertainty has been used. Specifically, given:

```
clause 1 : CF1 and
clause 2 : CF2 then
```

when we resolve we get

clause 3 : CF3 where

$CF3 = CF1 \times CF2$

However, the system has been implemented in a modular fashion so that it is very easy to test any number of different calculi, for example, one based on information theory as discussed in *[Mor87]*.

2 Future Work

In this section we outline some areas of future work.

2.1 Extension to Non-Horn Clauses

The present implementation is limited to horn clauses. The next step would be to allow the system to cater for non-horn clauses. However, this would be a major undertaking given the additional complexities associated with non-horn clauses.

2.2 Experimenting With Calculi

It is very easy, in our system, to test different calculi. It seems reasonable that different calculi may be appropriate to different domains. Work needs to be done in testing different calculi to determine for which domain or applications area they may be appropriate.

In a CFDL system the calculus module can be modified so that the Certainty Factor is a combination of several certainty factors. This issue could be explored in the future work.

2.3 Update in CFDL

As indicated by *[Pea90]* and discussed in section 4.2, the problem of consistency checking is difficult in NMR. This means that the problem of update in NMR systems will need to be looked at very closely.

Work needs to be done, in the CFDL system, to ensure that the KB is consistent when clauses are added or deleted.

2.4 Interacting Defaults

In the thesis we have not addressed the problem of interacting defaults. Work needs to be done to determine how this problem can be addressed in CFDL.

2.5 Application Areas

We have shown that our system works with some of the benchmark problems of NMR. Work needs to be done to determine which applications areas are appropriate for our system. In theory, our system should work for most application areas. However, many issues will need to be addressed when a system is implemented for a specific area.

3 Conclusions

The CFDL system addresses the problem of conflicting defaults in Default Logic and allows for the distinction between certain answers and those that are default or uncertain answers.

3.1 Conflicting Defaults

The problem of conflicting defaults in Default Logic is addressed in CFDL. The only way in which conflict in defaults can be resolved is if we have a greater degree of certainty in one of the defaults. This information would have been encoded in the certainty factor which we attach to each default. We use these certainty factors to resolve the conflicting defaults. For example, in the Republican/Quaker example we represent the KB as

hawk(X) $\leftarrow$ republican(X) : 0.9

dove(X) $\leftarrow$ quaker(X) : 0.95

republican(nixon) $\leftarrow$: 1.0

quaker(nixon) $\leftarrow$: 1.0

Since we have more certainty in the second default our conclusion will be "Nixon is a dove". This does not mean that a conflict cannot arise in our system. If we had

hawk(X) $\leftarrow$ republican(X) : 0.95

dove(X) $\leftarrow$ quaker(X) : 0.95

republican(nixon) $\leftarrow$: 1.0

quaker(nixon) $\leftarrow$: 1.0

there would be a conflict even in our representation. However, there is less chance of a conflict in our system.

3.2 More Than True/False Conclusions

The CFDL addresses the problem of the lack of distinction between "**there is reason to suppose P**" and "**P is true**" in Default Logic. We do this by using the information encoded in the certainty factors. Our resolution strategy allows us to find the most certain answer with its associated certainty factor. Hence we can do more than distinguish between statements of the form "**there is reason to suppose P**" and "**P is true**". We can also give the user the certainty factor with the answer. These certainty factors can be mapped to more English-like answers such as *very likely*, *unlikely* or *certain*.

For example, given the KB

flies(X) ← bird(X) : 0.9

bird(tweety) ← : 1.0

flies(chiree) ← : 1.0

bird(alf) ← : 0.4

we can conclude "***Chiree certainly flies***", "***Tweety most likely flies***" and "***It is unlikely Alf flies***".

4 Conclusion

In this chapter we have discussed some of the findings of our research and shown how the conclusions support our thesis statement.

BIBLIOGRAPHY: Thesis

Bre91. G. Brewka. Cumulative default logic: In defence of nonmonotonic inference rules. *Artificial Intelligence*, pages 183–205, 1991.

DP90. D. Dubois and H. Prade. An introduction to possibilistic and fuzzy logics. In *Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 742–762. Morgan Kaufmann Publishers, Inc., California, 1990.

Isr80. D. J. Israel. What's wrong with non-monotonic logic. In *Proc. of the First Annual National Conference On Artificial Intelligence*, pages 99–101. AAAI, 1980.

Lif89. V. Lifschitz. Benchmark problems for formal non-monotonic reasoning, version 2.00. In *Lecture Notes In Artificial Intelligence 346*, pages 202–218. Springer-Verlag, 1989.

McC80. J. McCarthy. Circumscription- a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

Mor87. J. Morrissey. *A Treatement of Imprecise Data and Uncertainty in Information Systems*. PhD thesis, University College Dublin, 1987.

Nut83. J. T. Nutter. What else is wrong with non-monotonic logics? In *Proc. Fifth Conference of the Cognitive Science Society*, pages 1–5, 1983.

Nut87. J. T. Nutter. Uncertainty and probability. In *Proc. Tenth IJCAI Milan 1987*, pages 373–379, 1987.

Par94. Viren Parasram. An approach to commonsense reasoning : Default logic augmented with certainty factors. Master's thesis, University of Windsor, 1994.

Pea90. J. Pearl. Probabilistic semantics for nonmonotonic reasoning: A survey. In *In Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 699–710. Morgan Kaufmann Publishers, Inc., California, 1990.

Poo88. D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

Poo92. D. Poole. Local users guide to theorist. Technical report, University of British Columbia, 1992.

Qui89a. *Quintus Prolog*, release 2.5 edition, 1989.

Qui89b. *Quintus ProWINDOWS*, release 1.1 edition, 1989.

RC81. R. Reiter and G. Criscuolo. On interacting defaults. In *Proc. of the Seventh International Join Conference On Artificial Intelligence*, pages 270–276. IJCAI, 1981.

Rei80. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

Rob65. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

SB75. E.H. Shortliffe and B.G. Buchanan. A model for inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379, 1975.

Sha76. G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey, 1976.

Sha90. G. Shafer. Chapter 7: Introduction. In *In Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 473–481. Morgan Kaufmann Publishers, Inc., California, 1990.

Sho76. E.H. Shortliffe. Mycin: Computer based medical consultation. *New York: American Elsevier*, 1976.

Zad65. L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

Zad78. L. Zadeh. Fuzzy sets as a basis for a theory of possibility. In *Fuzzy Sets and Systems*. North-Holland, Amsterdam, 1978.

Zad83. L. Zadeh. Commonsense knowledge representation based on fuzzy logic. *Computer*, 16(10):61–65, 1983.

APPENDIX A
ABBREVIATIONS

CF	Certainty Factor
CFDL	Certainty factor Default Logic
DL	Default Logic
MR	Monotonic Reasoning
NML	Non-Monotonic Logic
NMR	Non-Monotonic Reasoning

APPENDIX B
DEFINITIONS

Definition Default: A default is any expression of the form

$$\frac{\alpha(\mathrm{x}) : M\beta_1(\mathrm{x}), \ldots, M\beta_m(\mathrm{x})}{w(\mathrm{x})} \tag{28}$$

where $\alpha(\mathrm{x}), \beta_1(\mathrm{x}), \ldots, \beta_m, w(\mathrm{x})$

are well formed formulae (wff) whose free variables are among those of $\mathbf{x}=x_1, \ldots, x_n$. $\alpha(\mathbf{x})$ is called the prerequisite of the default, and $w(\mathbf{x})$ is its consequent.

Definition Monotonic logic:

A logic is said to be monotonic if and only if for any set of premises S and S^1:

$$S \subseteq S^1 \rightarrow \{A \mid S \vdash A\} \subseteq \{A \mid S^1 \vdash A\} \tag{29}$$

where $\vdash$ is the provability relation.

Definition Non-Monotonic logic:

A logic is said to be non-monotonic if and only if for any sets of premises S and S^1:

$$S \subseteq S^1 \not\rightarrow \{A \mid S \vdash A\} \subseteq \{A \mid S^1 \vdash A\} \tag{30}$$

That is if and only if its provability relation violates the property of monotonicity. Simply stated, any conclusion of S is not necessarily a conclusion of S^1.

APPENDIX C
NON-MONOTONIC REASONING : A SURVEY

The problem of reasoning when knowledge is incomplete requires a non-monotonic reasoning system since old conclusions may have to be retracted in the light of new knowledge. Researchers have approached this problem using either a computational or a formal approach. In the computational approach, attempts have been made to implement practical systems with non-monotonic properties. In the formal approach, a formal logic system is developed upon which the applications will be based. Many logic systems have been developed for this purpose (default logic, circumscription, etc.). However, no single logic system captures all aspects of incomplete knowledge. The debate continues among researchers as to whether there exists a 'non-monotonic logic' which can capture all aspects of non-monotonic reasoning.

This survey classifies various approaches to non-monotonic reasoning. Some of the important systems/formalisms in each class are examined and we also look at the application of non-monotonic reasoning to databases.

1 INTRODUCTION

1.1 Non-Monotonic reasoning ?

Non-monotonic reasoning is the type of reasoning performed when our information is incomplete. How we arrive at conclusions, given incomplete information, is the main topic of research in non-monotonic reasoning. A key feature of non-monotonic reasoning is that new information may invalidate previous knowledge and we may be forced to retract certain conclusions made earlier.

1.2 Why Non-Monotonic Reasoning ?

Non-monotonic reasoning is needed only

when knowledge is incomplete or inconsistent. The question is, can knowledge ever be complete and in general, the answer is no. However, in traditional data base systems (DBS), the knowledge explicitly represented in the database is treated as being complete and consistent. In such traditional DBS any new addition of facts only increases the number of conclusions and thus such a system will be monotonic.

The task of obtaining information from such traditional databases, though not trivial, did not require any great deal of intelligence. A key property of intelligence is flexibility. By this we refer to the ability of drawing conclusions and then retracting them if necessary in the face of new evidence. It is the retraction of the previous conclusions that is the basic idea in non-monotonic reasoning. If our computer programs are to act intelligently they will need to be similarly flexible.

Frost *[Fro86]* states that there are three types of circumstances in which non-monotonic reasoning may be appropriate:

- When knowledge is incomplete.
- When the universe of discourse is changing.
- In problem solving where temporary assumptions are made.

Uses of non-monotonic reasoning are discussed by Ginsberg *[Gin87]* . These include:

- Inheritance hierarchies which have very attractive computational properties.
- Closed-world databases in which negation is treated as "failure to prove."
- Reasoning about action in which the qualification problem and frame problem arises.
- Logic programming applications.

Computer users today expect the computer to possess some degree of intelligence. A proper understanding of non-monotonic reasoning is necessary if this expectation is to be realized. Present computer implementations of systems that show some degree of intelligence require large amounts of code and are not very efficient. The computational efficiency of non-monotonic reasoning systems however, must be viewed in the context of their increased effectiveness.

1.3 More Effective ?

Traditional reasoning systems were monotonic and as such there was no need to reexamine their previous conclusions in the light of new knowledge. Non-monotonic systems however, must re-examine the validity of previous conclusions, in the light of new knowledge, and retract those that are no longer applicable. Such systems will be more effective since they will have a deductive ability. The system, with the help of inference rules, will be able to deduce more conclusions from a given set of facts and revise its conclusions when needed. Thus a non-monotonic reasoning system will be much more effective than traditional reasoning systems.

If computers are to exhibit the common sense reasoning that we see in humans, then the study of non-monotonic reasoning is essential.

1.4 Layout

The structure of this survey is as follows. In section 1, we give an introduction to non-monotonic reasoning and discuss the need for non-monotonic reasoning systems. In section 2, we discuss monotonic and non-monotonic reasoning and give formal definitions of both. In section 3, we look at some classifications of non-monotonic reasoning systems and present our classification scheme. In section 4, we look at the computational approaches which include MICRO-PLANNER, TMS and inheritance hierarchies. In section 5, we look at the formal approaches which are further subdivided into closed-world and non closed-world approaches. The non closed-world approaches can be numeric or non-numeric. In section 6, we present some criticisms of non-monotonic reasoning. In section 7, we discuss the application of non-monotonic reasoning to databases. Here we explain what is a deductive database and explore some related aspects. In section 8, we look at the application of non-monotonic reasoning to deductive databases. In section 9, we discuss some of the findings of the survey. An article listings is given for all articles cited in the survey.

2 WHAT IS NON-MONOTONIC REASONING?

2.1 Monotonic reasoning

A logic is said to be monotonic if the addition of new premises (axioms) never invalidates old conclusions (theorems). The set of conclusions is said to increase monotonically with the set of premises.

Definition Monotonic logic:

A logic is said to be monotonic if and only if for any set of premises S and S^1:

$$S \subseteq S^1 - \{A \mid S \vdash A\} \subseteq \{A \mid S^1 \vdash A\}$$

where $\vdash$ is the provability relation.

(31)

This definition implies that any conclusion from S can never be invalidated in S^1. Traditional logics have always been monotonic. In his landmark paper McDermott *[MD80]* notes:

"Monotonic logics lack the phenomenon of new information leading to a revision of old conclusions."

However, in many instances, it is necessary to make conclusions with incomplete knowledge and it is possible that these conclusions may be invalidated when new knowledge comes to light.

2.2 Non-Monotonic reasoning

The type of reasoning, where previous conclusions have to be retracted in light of new knowledge is called non-monotonic reasoning.

Definition Non-Monotonic logic:

A logic is said to be non-monotonic if and only if for any sets of premises S and S^1:

$$S \subseteq S^1 \not\to \{A \mid S \vdash A\} \subseteq \{A \mid S^1 \vdash A\} \tag{32}$$

That is if and only if its provability relation violates the property of monotonicity. Simply stated, any conclusion of S is not necessarily a conclusion of S^1.

3 CLASSIFICATION OF NON-MONOTONIC REASONING SYSTEMS

3.1 Some Classifications

Several types of non-monotonic reasoning can be identified in the literature and several different approaches have been taken in classifying the non-monotonic reasoning systems.

Lukaszwicz *[Luk90]* classifies them as

1. modal non-monotonic logic,
2. default logic,
3. circumscriptive logics,
4. CWA logic,
5. some other formalisms

Brewka *[Bre91b]* classifies them as

1. modal approaches,
2. approaches based on non-monotonic inference rules,
3. circumscription,
4. preferred subtheories,
5. approaches based on conditionals.

Etherington *[Eth88]* on the other hand, gives the following classification:

1. closed world reasoning,
2. default or prototypical reasoning.

3.2 Our Classification

Our classification scheme firstly splits the approaches into computational and formal. The computational approaches (eg. MICRO-PLANNER) are those that attempt to implement non-monotonic reasoning in practical systems. The formal approaches (eg. Default Logic) are those that develop a formal logic system upon which the applications will be based. The formal approaches are then divided into closed world approaches (eg. Predicate Completion) and non-closed world approaches. The non-closed world approaches may be numeric (eg. Certainty Theory) or non-numeric (eg. Default Logic).

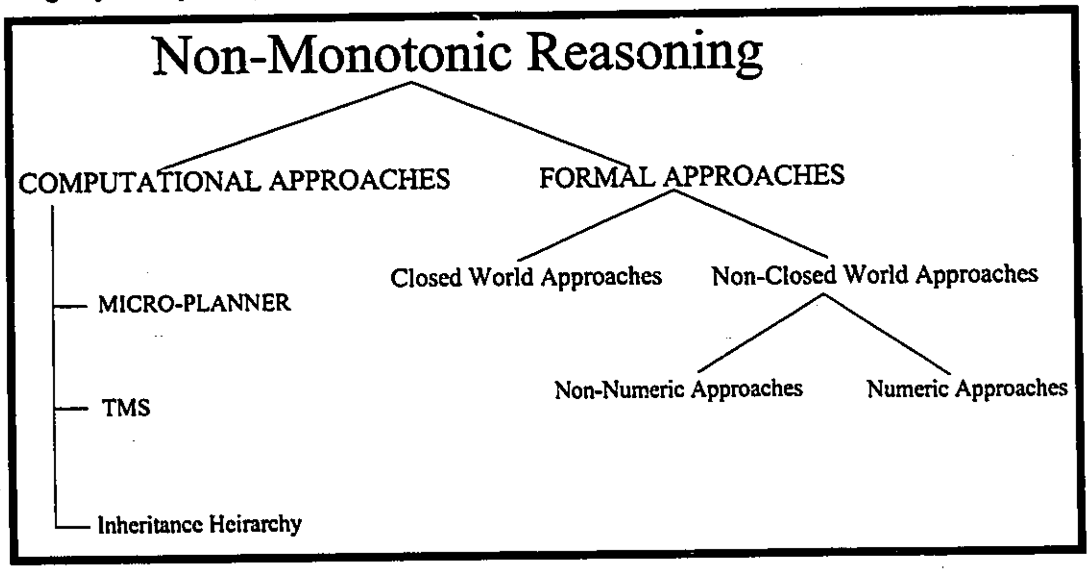


Figure 3.1 Approaches To Non-Monotonic Reasoning

4 COMPUTATIONAL APPROACHES

In the computational approaches, attempts have been made to implement practical systems with non-monotonic properties.

4.1 MICRO-PLANNER

One of the first such systems was MICRO-PLANNER *[SWC71]*. A MICRO-PLANNER program is a collection of subroutines called theorems which operate on a database. MICRO-PLANNER programs admit three types of subroutines called consequent, antecedent and erasing. Consequent subroutines are procedural representations of inference rules. Lukaszewicz *[Luk90]* gives an example of a consequent subroutine which represents 'All dogs are mammals' as

```
(THCONSEQUENT (Mammal ?x)
(THGOAL Dog(?x)))
```

which reads 'To prove an object, say x, is a mammal, try to prove that it is a dog'. The antecedent and erasing subroutines are used to handle the belief-revision problem. Lukaszewicz *[Luk90]* gives an example as

```
(THANTECEDENT (Innocent ?x)
(THERASE (Suspect ?x)))
```

which reads 'whenever a proposition of the form Innocent(x) enters the database, the proposition suspect(x), the argument of the erasing subroutine, if it exists, is to be removed from it'. To help the process of non-monotonic reasoning, MICRO-PLANNER employs a special operator called THNOT which is implemented as failure to prove and is the procedural analog of negation. This means THNOT(A) will succeed if the system's attempts to prove A fails.

The advantage of MICRO_PLANNER's system is its computational efficiency which is due to its specialized inference rules. The disadvantages are the lack of a general inference mechanism which forces the user to be responsible for specifying each inference scheme and the lack of formal semantics which makes the system difficult to predict in complex situations. *[Hay77]* gives more details on the latter disadvantage.

4.2 Truth-Maintenance Systems (TMS)

A second and important class of computational systems exhibiting non-monotonic behavior are truth-maintenance systems, also called belief revision systems or reason maintenance systems *[Doy79, deK86]* . The task of a TMS is to record and maintain beliefs.

Truth maintenance systems consist of a problem solver, a special database and a truth maintenance set of procedures (TMS). The problem solver depends on the TMS to provide it with the set of current beliefs, based on the special database. The problem solver uses its reasoning mechanism and the set of current beliefs to make new inferences. Any new inference may cause the set of beliefs to change. However, any new inference is not communicated directly to the TMS but rather to the special database. After updating the special database the problem solver passes control to the TMS. The TMS uses its own reasoning mechanism, distinct from that of the problem solver's, to realize a new set of current beliefs and then transfers control back to the problem solver.

Several approaches have been made to implement truth maintenance systems which have been organized in a variety of ways. However they all maintain the fundamental concepts of a TMS as outlined by Doyle *[Doy79]* in his landmark paper. The TMS Doyle outlines is a justification based TMS. In this approach, statements of beliefs are called nodes and each node is always in one of two states called 'in', if it is believed, and 'out' otherwise. A node also includes a set of justifications which consists of an ordered pair of lists called inlist and outlist. The inlist is a list of all the nodes which must be in the 'in' state and the outlist is a list of all the nodes that must be in the 'out' state if the specific belief (node) is to be 'in'. Whenever the problem solver applies its inference mechanism to compute some datum (consequent), this datum is added to the specialized database as a potential belief and the TMS is invoked. The TMS, as outlined before, is responsible for updating the current set of beliefs based on the updated special database.

The TMS provides two basic mechanisms to update the set of beliefs: truth maintenance and depedency-directed backtracking. Truth maintenance is used when a new justification is to be added or substracted from an existing belief (node). Depedency-directed backtracking is used if the current set of beliefs forces a contradictory node to be 'in'. A contradictory node is created by the problem solver when a new consequent creates a contradiction in the current state of the database.

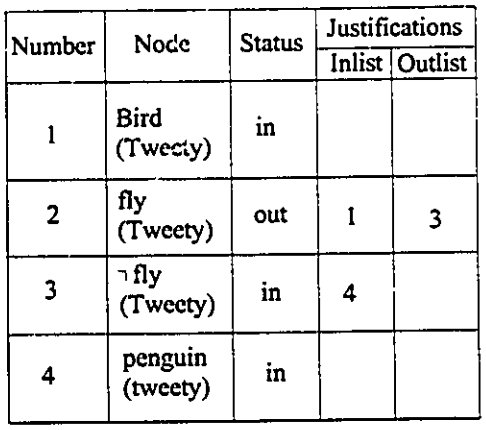
Strobel *[Str89]* gives an example with depedency-directed backtracking. He considers the following default theory:

$$\left\{ \frac{bird(x) : M fly(x)}{fly(x)}, \begin{array}{c} bird(Tweety)\ , \\ penguin(x) \rightarrow \neg fly(x) \end{array} \right\} \tag{33}$$

This theory states that if x is a bird and there is no information to the contrary then x flies, Tweety is a bird and if x is a penguin then x does not fly. The truth maintenance system would record this knowledge as in Figure 4.2.

Number	Node	Status	Justifications	
			Inlist	Outlist
1	Bird (Tweety)	in		
2	fly (Tweety)	in	1	3
3	¬ fly (Tweety)	out		
4	penguin (tweety)	out		

Figure 4.2 Example: TMS Belief Storage



Number	Node	Status	Justifications	
			Inlist	Outlist
1	Bird (Tweety)	in		
2	fly (Tweety)	out	1	3
3	¬ fly (Tweety)	in	4	
4	penguin (tweety)	in		

Figure 4.3 Example: TMS Beliefs after dependency-directed backtracking

This is interpreted as bird(Tweety) is currently believed and it is an assertion since it has an empty justification list. Fly(Tweety) is also believed ('in') since bird(Tweety) is 'in' and ¬fly(Tweety) is 'out'. The belief that Tweety is a penguin is 'out'.

If the theorem prover gets a datum that Tweety is a penguin then, it will enter this as a contradictory node into the special database, since penguin(Tweety) cannot be 'out' and 'in' at the same time. Control is transferred to the TMS which notes that there is an inconsistency since both fly(Tweety) and ¬fly(Tweety) are now derivable and therefore initiates dependency-directed backtracking. The set of beliefs at the end of the process will look like Figure 4.3.

Note that the new dependency that ¬fly(Tweety) is 'in' if penguin(Tweety) is 'in' has been added and the new set of beliefs has been generated.

DeKleer *[deK86]* noted several shortcomings in Doyle's TMS. One is the computational complexity of the algorithms used, especially the dependency-directed backtracking which is a very inefficient technique for ensuring consistency. Also, the TMS only allows one solution to be considered at a time (single-state problem). DeKleer therefore proposed a more efficient TMS called assumption based TMS (ATMS). In this method, each datum is labeled with the assumptions under which it holds, rather than the justifications. In contrast to the traditional truth maintenance systems the ATMS nodes have no states assigned to them. Lukaszwicz *[Luk90]* notes that instead with each node there is associated a table which (almost) explicitly represents the context under which the node holds. Therefore, the ATMS will always have to ask the user the current context and it is this which enables reasoning in multiple contexts. The advantages of this approach are that the overall database need not be consistent, dependency-directed backtracking can be avoided in most cases and the single-state problem can be overcome. However, the ATMS supports monotonic justifications only and thus never refers to what is disbelieved.

4.3 Inheritance Systems

This type of computational system with non-monotonic properties is based on semantic nets. Such systems combine the deductive structure of standard logic with non-standard reasoning facilities. Non-monotonic behavior is achieved by a uniform mechanism of preferring certain types of inferences.

The semantic network consists of nodes and links. The node may be a predicate node or a constant node. The links can be between any two nodes and they can be positive or negative. Lukaszewicz *[Luk90]* gives an example of a semantic net.

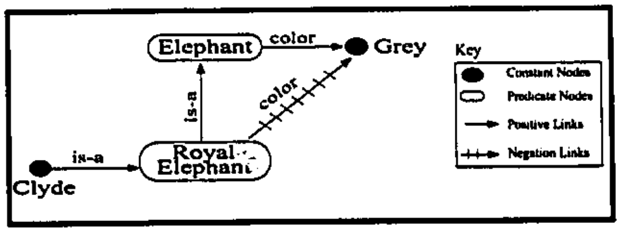


Figure 4.4 A Contradictory Semantic Network

Figure 4.4 has an example of a contradictory semantic network. The choice of whether Clyde is grey or not is dependant on the implementation decisions. In this example, if we choose the strategy of accepting the more specific information then Clyde is not grey. However, if we choose to accept the more general information then Clyde will be grey. The problem is that, in many systems, this selection strategy may not be fully specified. This is a major weakness of the system and as a result in an under-specified system its behavior may be difficult to predict.

Etherington *[ER83]* looks at a class of semantic networks called inheritance hierarchies with exceptions. He uses default logic to formalize the notion of inheritance systems with exceptions. The inheritance hierarchy consists of constant and predicate nodes together with assertion links (is-a links) and negation links (isn't-a links). The links are related to default rules and thus default logic can provide a semantics for inheritance systems with exceptions.

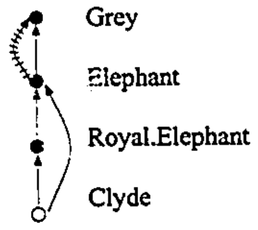


Figure 4.5 Inheritance Network with redundant statement

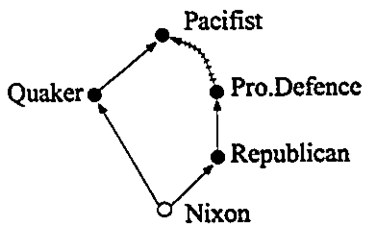


Figure 4.6 Ambiguous Inheritance Network

An early system NETL *[Fah79]* used the shortest path strategy to resolve ambiguities such as, 'what color is Clyde' in Figure 4.4. Using the shortest path strategy the answer would be Clyde is not grey since the path through the 'is-a' link is longer. However flaws in the shortest path ordering are pointed out by *[Tou84]*. Problems occur when handling true but redundant statements (Figure 4.5) and also in ambiguous inheritance networks (Figure 4.6). In figure 4.5, before the redundant statement , Clyde is an elephant, is entered NETL would deduce Clyde is not grey. However, after the redundant statement is added, NETL would now deduce that Clyde is grey. This is a problem since the addition of a redundant statement should not change the conclusion that a system makes. In figure 4.6, NETL would conclude that Nixon is a pacifist. However, whether Nixon is a pacifist or not is totally ambiguous and such a conclusion should not have been made.

Touretzky uses an inferential ordering method to order defaults. This ordering simplifies the representation of inheritance in default logics and solve the problems of redundant statements and ambiguous networks mentioned before.

Inheritance systems are still important despite the fact that default logic is a more powerful formalism. In fact, Touretzky notes:

"*Default logic is clearly a more powerful formalism than inheritance for representing knowledge, but the latter remains important due to its conceptual simplicity and efficient inference algorithms*".

5 FORMAL APPROACHES

5.1 Introduction

The formal approaches can be categorised as belonging to two basic types: closed world and non closed world. The non closed world approaches can be further divided into the numeric and non numeric types.

In the closed world approaches, the assumption is made that all relevant positive information is known and hence, all negative facts

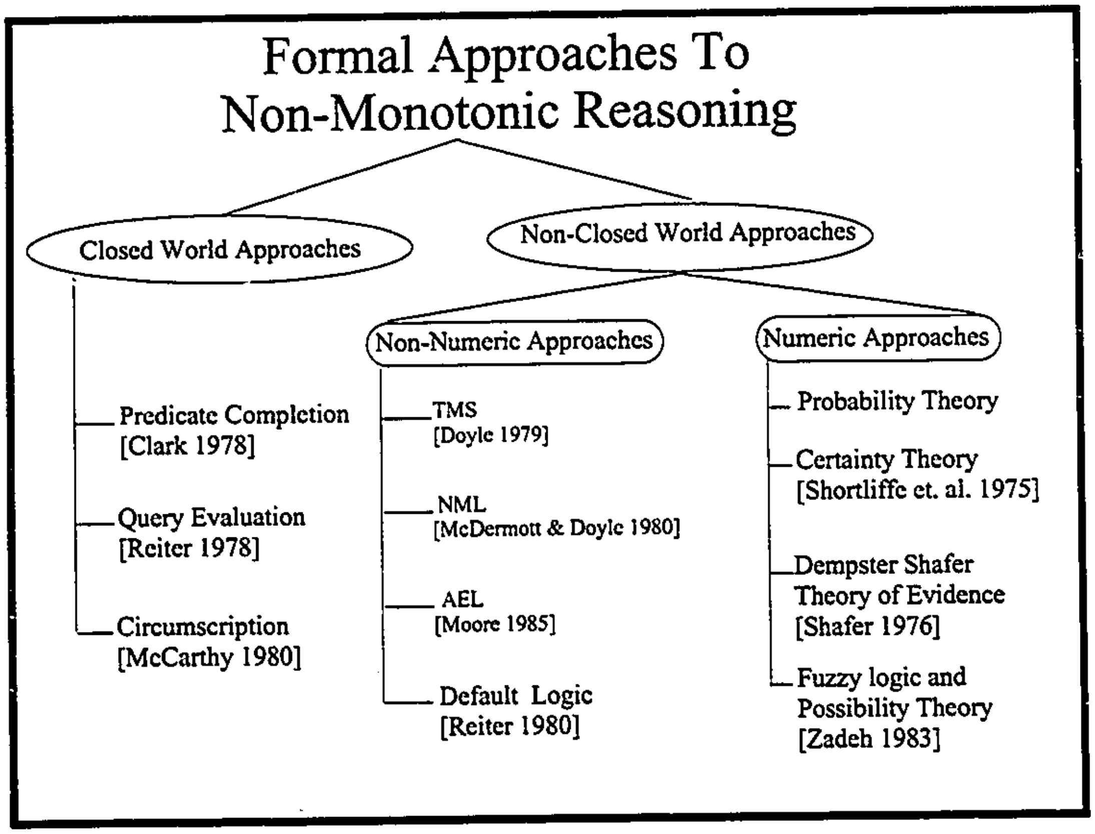

Formal Approaches To Non-Monotonic Reasoning
Closed World Approaches
Non-Closed World Approaches
Non-Numeric Approaches
Numeric Approaches
Predicate Completion [Clark 1978]
Query Evaluation [Reiter 1978]
Circumscription [McCarthy 1980]
TMS [Doyle 1979]
NML [McDermott & Doyle 1980]
AEL [Moore 1985]
Default Logic [Reiter 1980]
Probability Theory
Certainty Theory [Shortliffe et. al. 1975]
Dempster Shafer Theory of Evidence [Shafer 1976]
Fuzzy logic and Possibility Theory [Zadeh 1983]

Figure 5.7 Formal Approaches To Non-Monotonic Reasoning

need not be explicitly represented since they can be inferred from the absence of their positive counterparts. This approach thus, uses the closed world assumption (CWA). Some approaches in this category are predicate completion *[Cla78]* which uses a negation as failure rule (NAF), the CWA for query evaluation *[Rei78]* and circumscription *[McC80]*.

The second formal approach uses a non-closed world assumption. The assumption is that instead of assuming that whatever is unknown is false, attempts are made to fill the gaps in the knowledge. In the non-closed world approach two distinct classes can be identified: numeric and non-numeric classes. In the non-numeric class the gaps in the knowledge are filled by "default" or "prototypic" information. Included in this category are non-monotonic logic (NML) *[MD80]*, Autoepistemic Logic *[Moo85]*, TMS *[Doy79]* which is a formalization of the TMS discussed before in the computational approaches, and default logic *[Rei80]*. In the non-closed world numeric approach, statistical information is used to fill gaps in the knowledge, the idea being to obtain a statistical model for the data. The approaches of this type can be further grouped into four sub-classes: those that use Probability theory, those that use Certainty theory, those that use Dempster Shafer theory of evidence and those using Fuzzy logic and Possibility theory.

5.2 Closed World Approaches

Introduction CWA The closed world approach is motivated by the observation that the number of negative facts about a given domain is typically much greater than the number of positive ones. In the CWA *[Rei78]*, the assumption is made that all positive information has been specified and that any fact that is not specified or derivable from the information present is assumed to be false. In his landmark paper, Reiter *[Rei78]* notes

" *The implicit representation of negative facts presumes total knowledge about the domain being represented.* "

It can be argued that it is the lack of complete information that is the motivation for non-monotonic reasoning. Thus, how could a feasible solution be one that assumes all positive facts are known. Despite this, however, the CWA rule proves to be appropriate for most domains represented in typical databases. The opposite of CWA is the open world assumption (OWA). This assumes only the data explicitly represented in the database.

We illustrate how the CWA would be used in query evaluation using an example from *[Rei78]*.

| Teachers | = {a,b,c,d}
| Students | = {A,B,C}

Teach	
a	A
b	B
c	C
a	B

Figure 5.8 An Example to illustrate CWA evaluation of a Query

There are four teachers, (a,b,c,d), three students, (A,B,C), a teaches A and B, b teaches B and c teaches C. If you consider the query: "Who does not teach B?" Using the CWA we can argue the answer is those teachers in the set of teachers who do not teach B. The teachers who do not teach B is derived using the open world assumption and the result is subtracted from the set of teachers.

$$|\text{Teachers}| - || < x|\text{Teacher}|\text{Teach } x,B > ||_{OWA} \tag{34}$$

This gives the intuitive answer {c,d}.

Gallaire *[Gal81]* notes:

"... *when axioms are used, some inconsistencies may occur under CWA.*"

He illustrates this with the following example:

$$cat(x) \rightarrow black(x) \vee white(x) - - - axiom$$
$$with\ database:\ \{cat(felix)\} \tag{35}$$

The fact that Felix is black or white is not in the extensional data base (EDB), ie. explicitly represented, nor deducible from it. Therefore, using the CWA, we can deduce that Felix is not black or white, ie. ¬black(felix) and ¬white(felix) are derivable. However, this is inconsistent with the intensional database (IDB). Fortunately, no such problems arise when axioms have a restricted form called Horn Clause form but Horn Clause form is less expressive since indefinite knowledge cannot be represented. The axiom in the example was non-Horn and this led to the inconsistency.

Over the years many modifications have been proposed to Reiter's original CWA since its strong restriction that knowledge represented be complete cannot always be maintained. These modifications are less restrictive versions of the CWA rule in the sense that they allow safe reasoning with different degrees of incomplete knowledge. Lukaszewicz *[Luk90]* gives a good description of these modifications. He renames Reiter's original CWA as naive CWA (NCWA).

Definition Naive Closure *[Rei78]* : The naive closure of a theory T, denoted by NCWA(T), is the theory

$T \cup \{\neg A : T \nvdash A \; and \; A \in \text{HB}(T)\}$.

where HB is the Herbrand Universe

The set of theorems derivable from T by NCWA is identified with the set of all formulae classically derivable from NCWA(T). The requirement in NCWA is that knowledge be complete.

Lukaszewicz refers to Minker's *[Min82]* approach as generalized CWA (GCWA). The GCWA can be applied to incompletely specified worlds with disjunctions. For example, if we take W as a world with axioms:

$$\text{Student}(Viren) \vee \text{Teacher}(Viren) \quad (a)$$
$$\text{Course}(c_1) \wedge \text{Course}(c_2) \quad (b) \qquad (37)$$

Figure 5.9 Axioms specifying incomplete world

The disjunction makes the world incomplete and thus NCWA is not justified for W but the GCWA is justified and can be applied to conclude that Viren is not a course and c_1 and c_2 are neither students or teachers.

Definition Generalized closure *[Min82]* : The generalized closure of a theory T, denoted

GCWA(T), is the theory

$$T \cup \{\neg A : A \in NFREE(T)\}. \qquad (38)$$

We say that a formula B is derivable by GCWA from T if and only if $GCWA(T) \vdash B$. Here NFREE(T) denotes the set of all atoms from HB(T) which are free for negation in T. In the example, Figure 5.9

$$NFREE(W) = \left\{ \begin{array}{c} Course(Viren), Student(c_1), \\ Student(c_2), Teacher(c_1), Teacher(c_2) \end{array} \right\} \qquad (39)$$

The GCWA was further extended by Gelfond and Przymusinska *[GP86]* and referred to as careful CWA (CCWA). The new feature of the CCWA was that it allows us to restrict the effects of closing the world by specifying the predicate which may be affected by the CWA rule.

Definition CCWA *[GP86]* : Let T be a theory and suppose that **P**[*],**Q**[†] and **R**[‡] are disjoint tuples of all predicate constants occurring in T. A ground atom $A \in HB(T)$ is free for negation in T with respect to **P**,**Q** and **R** if and only if $A \in$ **P**$^{+}$[§]and there is no clause

$$C = C_1 \vee \cdots \vee C_n (n \geq 0),$$

where each C_i is in $\mathbf{P}^+ \cup \mathbf{R}^+ \cup \mathbf{R}^-$, such that

$$\begin{array}{ll} (i) & T \vdash A \vee C \\ (ii) & T \nvdash C. \end{array} \qquad (40)$$

We denote by NFREE(T;**P**;**Q**;**R**) the set of all atoms from **P**$^+$ which are free for negation in T with respect to **P**,**Q** and **R**.

Gelfond et al., *[GPP89]* give an approach called extended CWA (ECWA) which is im-

[*] **P** represents those aspects of the world which are to be closed.

[†] **Q** refers to the predicates which may be arbitrarily varied during the process of closure.

[‡] **R** includes the remaining predicate constants-those whose extensions must not be affected by the closure.

[§] **P**$^+$ (resp. **P**$^-$) is the set of all positive (resp. negative) ground literals constructible using predicate constants from **P** and function constants occurring in theory, T.

portant since it subsumes NCWA, GCWA and CCWA. ECWA augments the theory under consideration with ground sentences, rather than ground atoms.

Definition ECWA *[GPP89]* : Let T, **P**,**Q** and **R** be as defined in CCWA and suppose that A is an arbitrary ground sentence not involving predicate constants from **Q**. A is free for negation in T with respect to **P**,**Q** and **R** if and only if there is no clause

$$C = C_1 \vee \cdots \vee C_n (n \geq 0),$$

where each C_i is in $\mathbf{P}^+ \cup \mathbf{R}^+ \cup \mathbf{R}^-$, such that

$$(i) \quad T \vdash A \vee C$$

$$(ii) \quad T \nvdash C. \tag{41}$$

We denote by NFREE$_s$(T;**P**;**Q**;**R**) the set of all atoms from $\mathbf{P}^+$ which are free for negation in T wrt **P**,**Q** and **R**.

Predicate Completion In his landmark paper, Clark *[Cla78]* introduced the negation as failure (NAF) inference rule whereby $\neg$P can be inferred if every possible proof of P fails. This approach uses the CWA since if P cannot be proved then $\neg$P is assumed. The NAF rule is a weakened form of the CWA since it does not fully implement the "$\nvdash$" relation. In Clark's paper a Horn clause theorem prover is augmented with the NAF rule for dealing with negation in the query evaluation process of a logic database.

The NAF rule has been implemented in languages such as Prolog and PLANNER. In these implementations, because of the requirement of finite failure, the rule is sometimes called negation as finite failure (NAFF) and thus the syntactic form of the database as well as its logical content can play a role in what can be derived by NAF. Shepherdson [1984] illustrates this point with an example. If DB$_1$ = $\{P_a\}$ and DB$_2$ = $\{\neg P_a \rightarrow P_a\}$ then DB$_1$ and DB$_2$ are logically equivalent but Prolog can only prove P_a from DB_1. The attempt to prove P_a from DB_2 leads to an infinite proof tree since in order to prove P_a we must prove $\neg P_a$ and in order to prove $\neg P_a$ we must prove P_a. Hence NAFF will only work only with DB_1.

The most important contribution of Clark *[Cla78]* is to introduce a theory of completion which is more powerful than a first-order system augmented by NAF. The completion of a predicate requires the gathering together of all implied assumptions relevant to that predicate and making these assumptions explicit. In his paper Clark *[Cla78]* shows how the completed database (C(DB)) can be created by gathering together all the completion axioms for each relation of the DB. The advantage of the C(DB) is that the query evaluation process Clark defines will find every answer that is a logical consequence of the C(DB). Brewka *[Bre91b]* gives an example of a C(DB), which is the original DB, together with the completed axioms for each predicate.

The theory:

$$Bird(Tweety)\ (1)$$
$$\forall x.Penguin(x) \supset Bird(x)\ (2)$$
$$\forall x.Bird(x) \land \neg Penguin(x) \supset Flies(x)\ (3) \tag{42}$$

Implications about Bird:

$$\forall x.x = Tweety(x) \supset Bird(x)\ \left(1'\right) \tag{43}$$
$$\forall x.Penguin(x) \supset Bird(x)\ (2')$$

Completion Axioms:

$$Bird: \forall x.Bird(x) \supset$$
$$x = Tweety \lor Penguin\ (4)$$
$$Flies: \forall x.Flies(x) \supset \tag{44}$$
$$Bird(x) \land \neg Penguin(x)\ (5)$$
$$Penguin: \forall x.\neg Penguin(x)\ (6)$$

Figure 5.10 Example of Completion Axioms

The completion axiom (6) states that there are no penguins in the database. This conclusion is possible only if we know that the database explicitly or implicitly implies this. The completion of the predicate penguin gives this result which involves the deduction of all relevant details of that predicate from the complete database. Thus completion axioms convey information that is deductible from the complete database. Completion axiom (5) states that if x flies then x is a bird and x is not a penguin. Completion axiom (4) states that any ground atom in the database is either Tweety or a penguin.

Circumscription Circumscription was introduced as a form of non-monotonic reasoning by McCarthy*[McC80]* in his landmark paper. He states:

> *"Circumscription is a rule of conjecture that can be used by a person or program for 'jumping to certain conclusions' ".*

He also notes:

> *"The results of applying circumscription to a collection A of facts is a sentence schema that asserts that the only tuples satisfying a predicate P(x . . . z) are those whose doing so follows from the sentences of A."*

Lifschitz *[Lif91]* notes:

> *"The main idea of circumscription is to consider, instead of arbitrary models of a given axiom set, only the models that satisfy a certain minimality condition."*

Circumscription consists of adding new axioms (conjectures) to the underlying incomplete theory that force a minimal "closed-world" interpretation of particular aspects. Circumscription is not a non-monotonic logic but rather a form of non-monotonic reasoning augmenting ordinary first-order logic. Thus in circumscription, the non-monotonic theorems of a set of premises A, are defined to be the monotonic theorems of a certain superset $A \cup B$ of A. Circumscription is non-monotonic, in the sense that as A grows, the additional axioms change. McCarthy *[McC80]* gives an example which illustrates this point. If the sentence A is :

$isblock\ A \wedge\ isblock\ B \wedge\ isblock\ C$ (7)

then circumscribing isblock in (7) gives the schema (8) and on substitution we get (9).

$$\Phi(A) \wedge \Phi(B) \wedge \Phi(C) \wedge \forall x.(\Phi(x) \supset isblock\ x) \supset \forall x.(isblock\ x \supset \Phi(x)) \quad (8)$$

$$\forall x.(isblock\ x \supset (x = A \vee x = B \vee x = C)) \quad (9)$$

(45)

Figure 5.11 Circumscribing a Conjunction

If we adjoin isblock D to (7), we will no longer be able to infer (9), thus its non-monotonicity.

Circumscription tries to apply a minimization criterion to choose a preference from several possible conclusions. Ferreira *[FM89]* in his paper describes a method called "inscription" which is a dual of circumscription in the sense that it maximizes the ordering criterion amongst the possible conclusions. In this sense, circumscription tries to get the smallest number of conclusions which will be supported by known facts, whereas inscription tries to get the maximum number of conclusions that would be supported by the known facts.

Over the years, there have been various versions of circumscription proposed:

Domain circumscription *[McC80]*, previously called minimal inference, conjectures that the 'known' entities are all there are. The intension of domain circumscription is to syntactically formalize the domain closure assumption. One problem of the domain circumscription is that it is incapable of formalizing the unique name assumption *[Luk90]*.

Predicate circumscription *[McC80]* assumes that entities satisfy a given predicate only if they have to on the basis of a collection of facts. Predicate circumscription allows explicit completion axioms similar to Clark's completion axioms, to be conjectured as they are required. Predicate circumscription is a first-order axiom schema. McCarthy *[McC80]* gives an example of circumscribing a disjunction. Taking the disjunction (10) and circumscribing isblock (11) gives on substitution (12).

$$isblock\ A \vee isblock\ B \ (10)$$

$$(\Phi(A \vee \Phi(B))) \wedge \forall x.(\Phi(x) \supset isblock\ x) \supset$$
$$\forall x.(isblock\ x \supset \Phi(x)) \ (11)$$

$$\forall x.(isblock\ x \supset x = A) \vee$$
$$(isblock\ x \supset x = B) \quad (12) \qquad (46)$$

Figure 5.12 Circumscribing a Disjunction

In predicate circumscription only those predicates being minimized are allowed to vary.

Variable circumscription *[McC86]* is a generalization of predicate circumscription which allows certain other predicates to vary during minimization.

Formula circumscription *[McC86]* is an improved version of predicate and domain circumscription. It is a second order axiom schema and any predicate expression, rather than a simple predicate, may be minimized. Also, the predicates allowed to vary are no longer identified with those being minimized.

Certain predicates may be specified as variable.

Pointwise circumscription *[Lif88]* minimizes at one point at a time rather than everywhere simultaneously. In this approach a predicate P, may be True or False depending on the point at which minimization is being done. The advantage is different points can be defined with different priorities when minimizing P.

Perlis *[Per87]* looks at circumscribing with sets. He notes:

"There are very powerful and well-understood first-order set theories, going well beyond the power of second-order predicate logic. A first-order set theory with first-order (formula) circumscription gives at least all that second-order circumscription does. This includes allowing the use of a single formula instead of a schema."

Autocircumscription *[Per88]* is designed to isolate the features of determining what is (not) known to the agent itself, rather than what is (not) true in the outer world.

5.3 Non-Closed World Non-Numeric Approaches

Non-monotonic logic (NML) NML *[MD80]* is an example of a modal non-monotonic logic. It uses the modal operator M with the intuitive meaning "is consistent". McDermott *[MD80]* calls M a proposition-forming modality and notes:

"Informally, Mp is to mean that p is consistent with everything believed."

Moore *[Moo85]* gives an example with the modal operator, M. If the theory, T is:

$$Bird(Tweety) \qquad (13)$$

$$\forall x \begin{pmatrix} Bird(x) \wedge M(Can\text{-}Fly(x)) \\ \rightarrow Can\text{-}Fly(x) \end{pmatrix} \qquad (14)$$

(47)

If P = Can-Fly(x) then MP means that P is consistent with the non-monotonic theory that contains only the two axioms.

Fundamental to NML is the concept of a fixed point. Brewka *[Bre91b]* notes:

"The fixed points, intuitively, corresponds to belief sets which can be obtained by applying the standard inference rules of classical logics and throwing in as many formula of the form Mp as possible."

The fixed point consists of a set of all formulae which can be 'sanctioned' by the set of premises under an intended interpretation of M. Brewka *[Bre91b]* also states:

The problem with the fixed point approach is that in general the fixed points are hard to describe and difficult to find, since their definition is not constructive.

In the NML approach, if we have found the fixed points for a set of premises A, then the derivable formulae are defined as the intersection of all fixed points.

Definition Set of derivable formulae: Let A be a set of premises, FP(A) be the set of fixed points of A, then TH(A), the set of non-monotonic theorems of A, is the set of formulae

$$\{p \mid p \in S \forall S \in FP(A)\} \qquad (48)$$

McDermott and Doyle *[MD80]* noted that a major problem with NML was that the logic was too weak, in that the modal operator did not fully capture the meaning of consistency. The Yale shooting problem *[HMD86]* demonstrates that NML does not produce the expected results in all cases. The use of a frame axiom to solve the frame problem (i.e. the problem of finding an adequate representation of what does not change when an event occurs) leads to different results in the Yale shooting problem. The Yale shooting problem is a problem in the temporal domain. Hanks *[HMD86]* illustrates the problem. The problem involves a person, say Bob, a gun and three events: load, wait and shoot. In figure 5.13, at time s0, Bob is alive and the gun is empty. At s1, after the load event, Bob is still alive and the gun is now loaded. The problem is after the wait and shoot event is Bob alive or dead? In one model (Figure 5.14) Bob dies since it is considered not abnormal for the gun to remain loaded during the wait event and abnormal for Bob to live after the shoot event. In the other model (Figure 5.15) Bob lives since it is considered abnormal for the gun to remain loaded in the wait event and not abnormal for Bob to live after the shoot event. Hence, whether Bob lives or dies is now dependant on the choice of minimal models. The choice will therefore be dependant on the implementation strategy and this is not desirable.

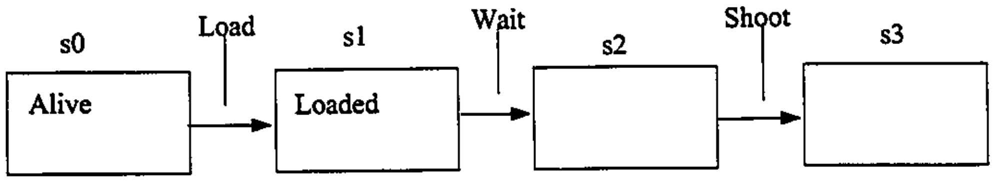
(a)


Figure 5.13 Yale Shooting Problem

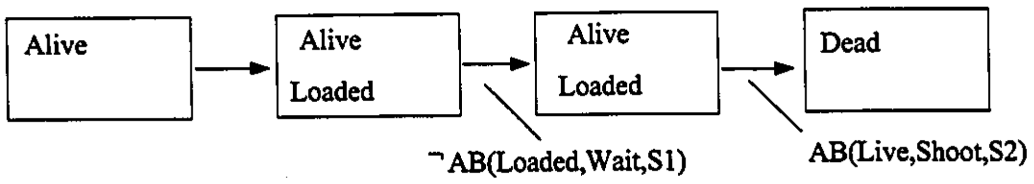
(b)


Figure 5.14 Yale Shooting Problem: Model 1

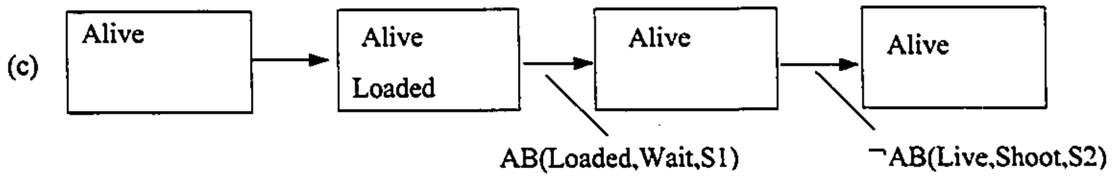


Figure 5.15 Yale Shooting Problem: Model 2

Autoepistemic logic (AEL) Moore *[Moo85]* proposed an improved version of NML called autoepistemic logic which characterizes autoepistemic reasoning. *[Moo85]* gives an example of the autoepistemic reasoning involved for believing "I do not have an older brother".

> *"I simply believe that if I did have an older brother I would know about it; therefore, since I don't know of any older brothers, I must not have any."*

AEL therefore, models the beliefs of agents who reflect on their own beliefs.

AEL has become by far, the most prominent modal non-monotonic logic. This is because it is a more accurate interpretation of NML *[MD80]*. McDermott and Doyle believed that the type of non-monotonic reasoning they were modelling was default reasoning and hence their interpretation of the NML axiom of the form:

$$\forall x\, Bird(x) \wedge M(can-Fly(x)) \supset Can{-}Fly(x) \tag{49}$$

would be "typical birds can fly". However Moore *[Moo85]* showed that NML in fact, interprets the axiom as "the only birds that do not fly are those known not to fly". Having made this distinction, Moore points out that default and autoepistemic reasoning are non-monotonic for different reasons. Default reasoning is tentative and thus defeasible. The non-monotonicity of autoepistemic statements enters due to their context-sensitive or indexical nature. This is since MP does not only mean that P is a theorem consistent with the non-monotonic theory but rather that it is consistent with the non-monotonic theory that has the specified number of axioms. Thus if the number of axioms ("context") changes then the autoepistemic statements will change.

AEL is a propositional modal logic. Moore argues that the possible sets of beliefs an ideally rational agent can hold based on a consistent set of premises A, are those sets T, such that

$$T = Th(A \cup \{LP \mid P \in T\} \cup \{\neg LP \mid P \notin T\}) \qquad (50)$$

L is the modal operator where LP means "P is believed". In words, the set, T consists of the theorems derivable from A, and the beliefs, P for which there is no reason to disbelieve P.

Default logic (DL) Default logic was originally introduced by Reiter *[Rei80]*. He addresses the problem of incomplete information by allowing new inference rules (defaults) to be added to a standard first-order logic. However unlike standard logic, the premises are allowed to refer to what is known and what is not known. The default rules are used for the completion of partial knowledge only when specific information is missing. For example, if there is specific information that Tweety flies, then there is no need to invoke the default rule: 'all birds fly except penguins'. However, if the only information we have is that Tweety is a penguin, then the default rule can be used to deduce that Tweety does not fly and this gives a basis for the completion of the partial knowledge. This differs from closed-world reasoning which employs a uniform completion strategy. In DL, the completion strategy can be modified by the use of different default rules.

Reiter *[Rei80]* gives the following definitions.

Definition Default: A default is any expression of the form

$$\frac{\alpha(\mathbf{x}) : M\beta_1(\mathbf{x}), \ldots, M\beta_m(\mathbf{x})}{w(\mathbf{x})} \qquad (51)$$

where $\alpha(\mathbf{x}), \beta_1(\mathbf{x}), \ldots, \beta_m, w(\mathbf{x})$ are well formed formulae (wff) whose free variables are among those of $\mathbf{x}=x_1, \ldots, x_n$. $\alpha(\mathbf{x})$ is called the prerequisite of the default, and w(x) is its consequent.

An example of a default rule is

$$\frac{Bird(x) : MCan{-}fly(x)}{Can{-}fly(x)} \qquad (52)$$

which can be interpreted as 'For every individual, x, if x is a bird and it is consistent to believe that x can fly, then it is believed that x can fly.' Thus in the absence of information to the contrary we will believe that if Tweety is a bird then she flies. *[Lif89]* gives different types of problems that involve default reasoning.

Definition Closed Default: A default is closed if none of $\alpha, \beta_1, \ldots, \beta_m$,w contains free variables.

Etherington *[Eth87]* notes that there are two main classes of defaults, normal and seminormal and virtually all of the defaults occurring in the literature fall into one of these two classes. Normal defaults are those with β(x) = w(x) while seminormal defaults are those with β(x) = w(x) or C(x) for some C(x). Of course, if the prerequisite is empty then the default may be taken to be any tautology.

Definition Default Theory: A default theory, Δ, is an ordered pair, (D,W), consisting of a set of defaults, D, and a set of first-order formulae, W.

The DL is a proof-based approach where the proof required is that of consistency. In DL a rule such as, "Birds fly", would be formalized as : "If x is a bird, and it is consistent that x can fly, then x does fly". This can be expressed using the formulation of Reiter as

$$\frac{b : f}{f} \tag{53}$$

where b denotes "Tweety is a bird" and f denotes "He can fly" and is read as: "If b is true, and it is consistent that f is true then assume f is true.

Etherington *[Eth87]* gives some new results which enhance the usefulness of Reiter's DL. Etherington shows the benefits of his approach by developing a theory of inheritance networks. The important contribution is that the notion of correct inference and sufficient conditions for the coherence of networks inference representations could be determined.

Brewka *[Bre91a]* notes that there are two problems with Reiter's default logic. The first being inconsistencies between justifications of non-normal defaults may lead to results that are not intuitive. *[RC81]* gives an example of defaults which may lead to such an unintuitive results. If we have the two defaults: "Typically high school drop outs are adults" and "Typically adults are employed" then we do not want to conclude that "Typically high school drop outs are employed". Thus, 'typically' is not necessarily transitive. The problem is what is to be done with conflicting defaults. *[RC81]* looks at interacting defaults and introduces the notion of a semi-normal default theory as a solution to conflicting defaults.

The second problem is that default logic is not cumulative. By this we mean that the addition of theorems to the set of premises may change the derivable formulas. Brewka proposes a cumulative default logic (CDL) as a solution to these two problems.

5.4 Non-Closed World Numeric Approaches

Introduction In a monotonic reasoning system conclusions made cannot be invalidated and hence there is no need for a degree of certainty to be associated with conclusions. However, in a non-monotonic reasoning system conclusions may have to be retracted when new information is available. Thus, it is reasonable to ask the question: 'How certain is a given conclusion ?'. The approaches discussed before do not address this question. For example, if we conclude that Tweety flies, using some default logic, there is no way of determining the certainty of this conclusion.

The numeric approaches attempt to deal with this problem. They assign certainty measures to conclusions. The mechanism for deriving conclusions can be implemented using if-then production rules. These rules can be of the form:

if <antecedent>

then <consequent> With Certainty X.

Morrissey *[Mor87]* notes:

"The certainty measure may be the probability that the antecedent implies the consequent or it may be a measure of some human expert's belief that the antecedent implies the consequent."

The problem is that these production rules are not independent of each other. They are interconnected and hence the difficulty lies in combining the certainty measures. Each type of numeric approach uses a different method to deal with this problem.

Subjectivist vs Frequentist interpretation of probability In the past, probability has been associated only with frequentism. This means that probability was regarded as a means to handle uncertainty only when frequency data was available. However, a subjective interpretation of probability has the advantage that it can be used even when frequency data is unavailable. Using a subjective interpretation we can establish probabilistic norms. Pearl *[Pea90b]* notes:

"The benefits for adopting probabilistic norms apply not only to syntactical approaches to non-monotonic reasoning, but also to semantical approaches such as those based on preferential models [Sho87]."

Probabilistic semantics for non-monotonic reasoning A survey on probabilistic semantics for non-monotonic reasoning was conducted by Pearl *[Pea90b]*. This survey points out several reasons for the use of numeric approaches:

1. The well established theoretical results provide shortcuts between the semantics and intended conclusions,

2. The conclusions are less subject to dispute,

3. They represent the empirical facts and not the particular reasoner's set of beliefs, and

4. Even if the purpose of default statements is to establish conversational conventions the process of formulating them should not ignore their empirical origin.

In such a setting the "birds fly" example would be represented as P [fly(x) | Bird(x)] = High and would read as "If x is a bird then x probably flies".

Four numeric approaches: probability theory, certainty theory, Dempster Shafer theory of evidence and fuzzy logic and possibilistic theory are discussed here.

Probability Theory Probability theory is the classical means of handling uncertainty. However, a prior knowledge of the individual events and the knowledge of how these events are related is required before an estimate of the probability of the various combinations of events can be determined. For example, if A and B are independent events then, the probability of A and B occurring is given by the product of the probabilities of A and B. That is

$$P(A \text{ and } B) = P(A) * P(B).$$

Similarly we have,

$$P(A \text{ or } B) = P(A) + P(B) — P(A \text{ and } B).$$

Another frequently used rule of probability theory is Bayes Rule. It allows you to estimate the probability of a hypothesis being true given certain related evidence. Using Bayes Rule an approach can be developed to implement non-monotonic reasoning. Shafer *[Sha90a]* notes:

> *"The Bayesians prefer to assess prior subjective probabilities for the different possible statistical models add then use the data to update these prior probabilities to posterior probabilities, while the frequentist prefer to rely on the data alone to estimate the model."*

Pearl *[Pea90a]* notes:

> *". . . the essence Bayesianism - to postulate the existence of probabilities we do not really have to, assess their magnitude, complete the model, draw conclusions from that model, and check whether the conclusion is highly sensitive to the assumptions."*

The problem faced by such systems are that the interrelations must be clearly understood and probability measures updated, whenever new data is added, for the conclusions to be reliable.

Certainty Theory Certainty theory *[SB75]* is, in effect, an approximation to probability theory, but it uses incomplete knowledge *[Mor87]*. The theory was developed in an attempt to model the inexact reasoning processes of medical experts and was implemented as part of the MYCIN expert system *[Sho76]*. The approach taken is to maintain two values for each rule in the system:

- MB[h,e] = X

  is the measure of the increased belief in hypothesis h given evidence e.

- MD[h,e] = Y

  is the measure of the increased disbelief in hypothesis h given evidence e.

These measures of belief and disbelief correspond to subjective estimates given by experts. They are related to the Probability theory in the following way:

$$MB[h,e] = \left\{ \begin{array}{c} 1\ iff\ P(h) = 1 \\ \frac{max|P(h|e)| - P(h)}{max|1,0| - P(h)}\ otherwise \end{array} \right\} \tag{54}$$

$$MD[h,e] = \left\{ \begin{array}{c} 1\ iff\ P(h) = 0 \\ \frac{min|P(h|e)| - P(h)}{max|1,0| - P(h)}\ otherwise \end{array} \right\} \tag{55}$$

where P(h) is the expert's subjective belief in hypothesis h, and P(h|e) is the experts's subjective belief in hypothesis h given evidence e.

A third measure is the certainty factor which combines the measures of belief and disbelief and is defined as:

CF[h,e] = MB[h,e] — MD[h,e]

The certainty factor is used to rank competing hypotheses. Its advantage is that if exact probabilities cannot be obtained then it can provide a subjective estimate.

Dempster Shafer Theory of Evidence

This theory *[Sha76]* sets up belief functions over sets of objects. Morrissey *[Mor87]* notes:

"A belief function associates a number between zero and one with a proposition. The number indicates the degree of belief in the proposition given

the evidence. The theory concentrates on combining degrees of belief given different bodies of evidence. It is not concerned with how the numbers are determined."

Shafer *[Sha90b]* notes:

" *The theory of belief functions is based on two ideas: the idea of obtaining degrees of belief for one question from subjective probabilities for a related question and Dempster's rule for combining such degrees of belief when they are based on independent items of evidence."*

The problem with Dempster Shafer's theory of evidence is that it is difficult to implement.

Fuzzy Logic and Possibility Theory

Dubois and Prade *[DP90]* note:

"The expression 'fuzzy logic' is used to refer to a variety of approaches proposing a logical treatment of imperfect knowledge usually referring explicitly to fuzzy-set theory. However, a distinction among these approaches can be made between those that deal primarily with vagueness and those whose primary concern is uncertainty"

Fuzzy logic *[Zad83]* is used to present certain statements which other logics can not handle. For example, the statement: 'If a car which is being offered for sale is old and cheap then it is probably not in good condition' is rather vague and uncertain and involves a degree of truth *[Mor87]*.

Possibility theory *[Zad78]* is based on earlier work on Fuzzy Set Theory *[Zad65]*. The work in this area has been continued by Dubois and Prade. They discuss a possibilistic logic *[DP90]*, which is a logic of partial ignorance and note that the possibility theory captures, in a very simple way, states of knowledge ranging from complete information to total ignorance.

6 CRITICS OF NON-MONOTONIC LOGICS

Israel *[Isr80]* states that "logic is, by its very definition, monotonic, and the notion of "non-monotonic logic" is a contradiction in terms." Israel states that the formal approaches to non-monotonic inferences are in general, non semi-decidable. Non-monotonic reasoning systems are very slow because of the repeated need for consistency checking. There is also the multiple extension problem which arises in a situation in which having applied one default rule we can not apply another.

Nutter *[Nut83]* argues that the problem with non-monotonic approaches is that they have not distinguished between the guarded statements of the form "There is reason to suppose that P" and statements of the form "P". If this is done the "There is reason to suppose that P" and "¬P" do not contradict and hence there is no need for retraction. If the problem of incomplete knowledge is approached in this way then a monotonic logic would be sufficient and there would be no need for a non-monotonic logic. She further argues that if the distinction is not made, valuable information will be lost. Nutter notes two cases:

"(a) the system will not know and be able to report the difference between those conclusions which its premises warrant without reservation and those conclusions which its premises only suggest, and (b) because the system will lose access to the reasonable assumptions when specific information overrides them, it will be unable to detect and state that a reasonable ex-

pectation has failed."

The probabilists argue that since probability theory is much better understood than non-numeric methods, it should be used to implement non-monotonic inference.

Despite these criticisms, the current research effort in non-monotonic logics is very active, both in the theoretical aspects and practical applications.

7 APPLICATION OF NON-MONOTONIC REASONING TO DATABASES

7.1 Introduction

Traditional database systems are restrictive since they require complete information and are limited to storing facts. The information is completed using the CWA. Deductive database management systems (DDBMS) are more flexible in that they allow information and rules for the deduction of new information. By their very nature deductive databases are non-monotonic and a good application area for non-monotonic reasoning.

Logic has been used to formalize database concepts. An excellent survey of logic as it is applied to databases is given in Gallaire et. al. *[GMN84]*. The paper aims "... to show that logic provides a convenient formalism for studying classical database problems". It looks at the applications to databases, where logic may be used both as an inference system and as a representation language. Logic provides a convenient formalism for traditional as well as deductive databases. Kowalski *[Kow78]* notes:

> "*... logic is useful both for conventional databases and programs as well as for the computational data bases which lie between the extremes.*" .

The extremes being deterministic programs at one end and explicitly described databases at the other. Frost *[Fro86]* notes

> *"since the semantics of logic is well-defined, if a logical counterpart to a database concept can be found then the semantics of that concept may also be well-defined".*

It is very likely that a non-monotonic logic would be useful in the formalizing of deductive database concepts. The traditional databases are a subset of deductive databases and therefore such a formalism, based on non-monotonic reasoning, will be very useful.

7.2 Deductive Databases Non-Monotonic ?

In a deductive database, inference rules are present which allow new facts to be deduced from those explicitly stored. Hence, in some cases, it may be easier to implement the required intensional database (IDB) in terms of a deductive database rather than the traditional extensional database (EDB). Gallaire et al. *[GMN84]* note:

> *"... the facts in the EDB may be viewed as extensions of the intensional axioms in the IDB."*

A formal definition of deductive databases is given in *[GMN84]*.

The deductive mechanism is rule driven and the number of conclusions deducable may decrease when a new fact is added. For example, we may be able to deduce that Tweety is a bird, Tweety flies, Tweety has feathers etc. However, if the fact that Tweety is a penguin is entered into the database then many of the previous conclusions will no longer be deducable and hence the non-monotonicity of deductive databases.

Deductive databases may be thought of as "logic databases", since they are based on logic, and knowledge bases (KB), since they can be thought of as containing knowledge as opposed to facts only, but essentially they are the same.

7.3 Choice of Programming Paradigm for Deductive Databases

Waugh et al. *[Wo90]* notes:

"The deductive database model is seen as a natural progression from :he relational model; the query language should reflect this."

They designed SQUIRREL, which is an extended form of SQL, as a query language for a Prolog-based deductive system. Note here the use of the logic programming paradigm for implementing a deductive database. Many applications use a logic programming paradigm. Giovannetti et al. *[GLMP91]* propose Kernel-LEAF, a logic plus functional language which tries to combine the advantages of both paradigms. The object oriented paradigm is also being used to implement databases and could be used in the future for deductive databases. However, at present, the logic programming paradigm seems to be the most useful.

7.4 Query Evaluation in Deductive Databases

In traditional databases, query evaluation involves a search of the facts in the database. Relation algebraic operations are also used to help evaluate queries. In a deductive database, for query evaluation, noι only the facts explicitly represented must be taken into account, but also those facts that are "deducible" from them and thus a deductive mechanism is required. The mechanism often used is a 'refutation' technique based on resolution.

Green *[Gre69]* gives a working computer program, QA3, which uses a resolution-type theorem-prover as its deductive mechanism to implement a question answering system.

8 NON-MONOTONIC REASONING AND DEDUCTIVE DATABASES

The fact that non-monotonic logic can be implemented in deductive databases has already been shown. The question is, what approach should be taken. Clark *[Cla78]* suggests the use of NAF meta rule and in fact, it is this rule that is implemented in Prolog as NAFF. The problem with NAF is that the query evaluation process in general is not complete and the only way to ensure that every solution to a query is found is to impose constraints on the database and its queries. In Prolog, one limitation is that only horn clauses be used and this greatly inhibits the type of knowledge that can be represented. In particular, disjunctions and negations cannot be used.

Reiter *[Rei78]* in his paper looks at the use of the CWA in deductive question answering systems. However this naive CWA is only applicable to Horn databases. In his paper Minker *[Min82]* extends the CWA to a GCWA which is applicable to both Horn and non-Horn databases.

Bossu and Siegel *[BS85]* identify a system of non-monotonic reasoning called subimplication. They *[BS85]* note:

"... *it is possible to set up a logical model of databases using subimplication and two sets of stable formulas. This model allows the construction of a system for creating, updating and querying databases that is totally transparent to the user.*"

9 DISCUSSION

Research in non-monotonic reasoning has been very active in the last decade and continues to be so in this decade. The motivation is to find a formal theory for everyday common sense reasoning. The many different formalisms that have been proposed all seem to capture only some aspects of common sense reasoning. The problems of reasoning about action and time as illustrated by the Yale shooting problem cannot be solved by any of the formalisms in the survey. Some of the formalisms are related, for example, circumscription and NAF *[GPP89]*, Default logic and AEL *[Kon88]*. However, there is no single formalism that can be regarded as "the" formal theory of common sense reasoning.

Two approaches has been identified for further research. The first, focuses on a computational approach where available formalisms are used to represent parts of the knowledge of realistic domains and to test whether systems based on these representations work as expected. In the second approach, the problems of computation are left to be dealt with after "the" non- monotonic logic is found. Brewka *[Bre91b]* suggests:

"There will probably not be much progress in the development of formalizations, nor any increase in the trust in the existing ones, without programs which handle more realistic examples that those that have been studied so far."

The logic programming language, Prolog, has been the choice for many implementations. Prolog has been used to implement new systems such as SQUIRREL: an extended SQL for a deductive database system *[Wo90]* , SATCHMO: a theorem prover *[MB88]* that is being used in the Japanese fifth generation project.

General application areas of non-monotonic reasoning are given in *[McC86]*. Strobel *[Str89]* cites the following application areas:

deductive databases, logic programming, diagnosis, reasoning about action and natural language processing.

The resolution principle *[Rob65]* was a landmark which made automated deduction systems practical. Such systems have been adapted for logic programming. Prolog can be used to implement deductive database systems. However, much work needs to be done on the integration of resolution and non-monotonic reasoning. This will form a topic in my thesis.

Much remains to be done in deductive databases and other application areas and also in the finding of "the" non-monotonic logic. However, due to the need for common sense reasoning systems, the research in the area will continue to be active. I believe that finding "the" non-monotonic logic, if at all possible, is in the far future. However, the need and the time for practical application programs is here.

10 REFERENCES

References for the NMR Survey can be found in the next appendix.

APPENDIX D

BIBLIOGRAPHY of NMR

AG91. S. Abiteboul and S. Grumbach. A rule-based language with functions and sets. *ACM Transactions on Database Systems*, 16(1):1–30, 1991.

AV91. S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.

Azz88. H. Azzoune. Type inference in prolog. In *9th International Conference On Automated Deduction*, pages 258–277. CADE, 1988.

BDP$^+$88. C. Beierle, J. Dorre, U. Pletat, C. Rollinger, P. H. Schmitt, and R. Studer. The knowledge representation language l_{lilog}. In *Lecture Notes In Computer Science 385,CSL 88*, pages 14–49. Springer-Verlag, 1988.

Bel90. John Bell. The logic of nonmonotonicity. *Artificial Intelligence*, 41:365–374, 1989/90.

BH89. Nicole Bidiot and Richard Hull. Minimalism, justification and non-monotonicity in deductive databases. *Journal of Computer and System Sciences*, 38:290–325, 1989.

BLO88. R. M. Butler, R. Loganantharaj, and R. Olson. Notes on prolog program transformations, prolog style, and efficient compilation to the warren abstract machine. In *9th International Conference On Automated Deduction*, pages 323–332. CADE, 1988.

BLS88. H. K. Büning, U. Löwen, and S. Schmitgen. Loop detection in propositional prolog programs. In *Lecture Notes In Computer Science 385, CSL 88*, pages 148–165. Springer-Verlag, 1988.

BM79. R. S. Boyer and J. S. Moore. *A Computational Logic*. Acadamic Press, Inc., 1979.

BMM89. P. Besnard, Y. Moinard, and R. E. Mercer. The importance of open and recursive circumscription. *Artificial Intelligence*, 39:251–262, 1989.

Boy71. R.S. Boyer. *Locking: a Restriction of Resolution*. PhD thesis, University of Texas at Austin, 1971.

Bra90. Ivan Bratko. *PROLOG Programming for Artificial Intelligence*. Addison - Wesley Publishing Company, second edition, 1990.

Bre91a. G. Brewka. Cumulative default logic: In defence of nonmonotonic inference rules. *Artificial Intelligence*, pages 183–205, 1991.

Bre91b. G. Brewka. *Nonmonotonic Reasoning: Logical Foundations Of Common Sense*. Cambridge University Press, 1991.

BS84. G. Bossu and P. Siegel. Nonmonotonic reasoning and databases. In *Advances in Data Base Theory Vol 2 ed. Gallaire, H. and Minker, J and Nicolas, J. M.*, pages 239–284. Plenum Press New York and London, 1984.

BS85. G. Bossu and P. Siegel. Saturation, nonmonotonic reasoning and the closed- world assumption. *Artificial Intelligence*, 25:13–63, 1985.

BS88. P. Besnard and P. Siegel. Supposition-based logic for automated nonmonotonic reasoning. In *9th International Conference On Automated Deduc-*

tion, pages 592–601. CADE, 1988.

BST89. Henri E. Bal, Jennifer G. Steiner, and Andrew S. Tannenbaum. Programming languages for distributed computing systems. *ACM Computing Surveys*, pages 261–322, 1989.

CH91. V. Chandru and J. N. Hooker. Extended horn sets in propositional logics. *Journal of the ACM*, 38(1):205–221, 1991.

Cho86. D.N. Chorafas. *Fourth and Fifth Generation Programming Languages, Vol 1*, chapter Expert Systems and Knowledgebanks, pages 189–200. Mc Graw Hill, New York, 1986.

CL73. C.L. Chang and R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York and London, 1973.

Cla78. K.L. Clark. Negation as failure. In *Logic and Data Bases ed. Gallaire, H. and Minker, J.*, pages 293–321. Plenum, New York, 1978.

Dat90. C. J. Date. *An Introduction To Database Systems*, volume 1. Addison-Wesley Publishing Company Inc., 1990.

Dav80. M. Davis. The mathematics of non-monotonic reasoning. *Artificial Intelligence*, 13:73–80, 1980.

deK86. Johan deKleer. An assumption based tms. *Artificial Intelligence*, 28:127–162, 1986.

Del88. J. P. Delgrande. An approach to default reasoning based on a first- order conditional logic: Revised report. *Artificial Intelligence*, 36:63–90, 1988.

DFP86. J. Darlington, A.J. Field, and H. Pull. The unification of functional and logic languages. In *Logic Programming Functions, Relations and*

Equations ed. DeGroot, D. and Lindstrom, G., pages 37–70. Prentice-Hall, 1986.

dK89. Johan deKleer and K. Konolige. Eliminating the fixed predicates from a circumscription. *Artificial Intelligence*, 39:391–398, 1989.

DM88. M. Danelutto and A. Macini. A temporal logic approach to specify and to prove properties of finite state concurrent systems. In *Lecture Notes In Computer Science 385,CSL 88*, pages 63–79. Springer-Verlag, 1988.

Doy79. J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

DP90. D. Dubois and H. Prade. An introduction to possibilistic and fuzzy logics. In *Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 742–762. Morgan Kaufmann Publishers, Inc., California, 1990.

DP91. D. Dubois and H. Prade. Epistemic entrenchment and possibilistic logic. *Artificial Intelligence*, 50:223–239, 1991.

Dra88. Nikos Drakos. Reason maintenance in horn-clause logic programs. In *Reason Maintenance Systems and Their Applications ed. Smith, B. and Kelleher K.*, pages 77–93. John Wiley and Sons, 1988.

DW91. Jon Doyle and W. P. Wellman. Impediments to universal preference-based default theories. *Artificial Intelligence*, 49:97–128, 1991.

EIG88. G. Escalada-Imaz and M. Ghallab. A practically efficient and almost linear unification algorithm. *Artificial Intelligence*, 36:249–263, 1988.

EMR85. D. W. Etherington, R. E. Mercer, and R. Reiter. On the adequacy of predicate circumscription for closed world reasoning. *Computational*

Intelligence, 1:11–15, 1985.

EOP91. N. Eisinger, H. J. Ohlbach, and A. Pracklein. Reduction rules for resolution-based systems. *Artificial Intelligence*, 50:141–181, 1991.

ER83. D. W. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *Proc. of the Third National Conference On Artificial Intelligence*, pages 104–108. AAAI, 1983.

Eth87. D. W. Etherington. Formalizing nonmonotonic reasoning systems. *Artificial Intelligence*, 31:41–85, 1987.

Eth88. D. W. Etherington. *Reasoning with Incomplete Information*. Pitman London/ Morgan Kufmann Publishers Inc., 1988.

Fah79. S. E. Fahlman. *NETL: A System for Representing and Using Real World Knowledge*. MIT Press, Cambridge, MA., 1979.

Far88. H. Farreny. *AI and EXPERTISE heuristic search, inference engines, automatic proving*, chapter 3 Automatic theorem proving, pages 158–160. John Wiley and Sons, 1988.

FH88. M. Franzen and L. J. Henschen. A new approach to universal unification and it's application to ac unification. In *9th International Conference On Automated Deduction*, pages 642–657. CADE, 1988.

FM89. C.P. Ferreira and J. P. Martins. Inscription- a rule of conjecture. In *Proc. 4th Portugese Conference on Artificial Intelligence*, pages 141–150. EPIA, 1989.

Fro86. R. Frost. *Introduction to Knowledge Base Systems*. McGraw-Hill Publishing Company, 1986.

Gal81. H. Gallaire. Impacts of logic on data bases. In *Proc. Very Large Data Bases*, pages 248–257. IEEE, 1981.

Gal86. J.H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row New York, 1986.

GG88. S. D. Goodwin and R. G. Goebel. Non monotonic reasoning in temporal domains: The knowledge independence problem. In *Lecture Notes In Artificial Intelligence 346, Non-Monotonic Reasoning*, pages 187–201. Springer-Verlag, 1988.

Gin87. M. L. Ginsberg. *Readings in Non Monotonic Reasoning*, chapter Introduction, pages 1–19. Morgan Kaufmann Publishers Inc., 1987.

Gin89. M. L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

GLMP91. E. Giobannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel-leaf: A logic plus functional language. *Journal of Computer and System Sciences*, 42:139–185, 1991.

GMN78. H. Gallaire, J. Minker, and J.M. Nicolas. An overview and introduction to logic and data bases. In *Logic and Data Bases ed. Gallaire, H. and Minker, J.*, pages 3–30. Plenum, New York, 1978.

GMN84. H. Gallaire, J. Minker, and J.M. Nicholas. Logic and databases: A deductive approach. *ACM Computing Surveys*, pages 231–246, 1984.

GP86. M. Gelfond and H. Przymusinska. Negation as failure: Careful closure procedure. *Artificial Intelligence*, 30:273–287, 1986.

GPP89. M. Gelfond, H. Przymusinska, and T. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:75–94, 1989.

Gra84. P. M. D. Gray. *Logic, Algebra and Data Bases*, chapter Representing Programs by Clauses: Prolog, pages 73–96. Ellis Horwood Ltd., 1984.

Gre69. C. C. Green. Theorem proving by resolution as a basis for question-answering systems. In *Machine Intelligence 4 ed. Meltzer, B and Michie, d.*, pages 183–205. Edinburgh University Press, 1969.

Hay77. P. J. Hayes. In defence of logic. In *Proc. 5th IJCAI Cambridge, MA*, pages 559–565, 1977.

HM90. J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

HMD86. S Hanks and D. Mc Dermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proc. of the Fifth National Conference On Artificial Intelligence*, pages 328–333. AAAI, 1986.

HT88. M. Hoshida and M. Tokoro. Alex: The logic programming language with explicit control and without cut-operators. In *Lecture Notes In Artificial Intelligence 383, Logic Programming 88*, pages 82–95. Springer-Verlag, 1988.

Imi84. T. Imielinski. On algebraic query processing in logical databases. In *Advances in Data Base Theory Vol 2 ed. Gallaire, H. and Minker, J and Nicolas, J. M.*, pages 285–318. Plenum Press New York and London, 1984.

Imi87. T. Imielinski. Results on translating defaults to circumscription. *Artificial*

Intelligence, 32:131–146, 1987.

Int90. Interface Computer GmbH. *Ifprolog Manual*, version 4.0 edition, 1990.

Isr80. D. J. Israel. What's wrong with non-monotonic logic. In *Proc. of the First Annual National Conference On Artificial Intelligence*, pages 99–101. AAAI, 1980.

IW91. Y. E. Ioannidis and E. Wong. Towards an algebraic theory of recursion. *Journal of the ACM*, 38(2):111–999, 1991.

Kon88. K. Konolige. On the relation between default and autoepistemic logic. *Artificial Intelligence*, 35:343–382, 1988.

Kor85. R. E. Korf. Depth-first iterative-deepening:an optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

Kow68. P. J. Kowalski, R. aand Hayes. Semantic trees in automatic theorem proving. In *Machine Intelligence 3 ed. Meltzer, B. and Michie, d.*, pages 87–101. Edinburgh University Press, 1968.

Kow78. R. Kowalski. Logic for data description. In *Logic and Data Bases ed. Gallaire, H. and Minker, J.*, pages 77–103. Plenum, New York, 1978.

KP90a. P. G. Kolaitis and C. H. Papadimitriou. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):1–14, 1990.

KP90b. P. G. Kolaitis and C. H. Papadimitriou. Some computational aspects of circumscription. *Journal of the ACM*, 37(1):1–14, 1990.

KP91. P. G. Kolaitis and C. H. Papadimitriou. Why not negation by fixpoint? *Journal of Computer and System Sciences*, 43:125–144, 1991.

Kup90. Gabriel M. Kuper. Logic programming with sets. *Journal of Computer and System Sciences*, 41:44–64, 1990.

Lif85. V. Lifschitz. Closed-world databases and circumscription. *Artificial Intelligence*, 27:229–235, 1985.

Lif87a. V. Lifschitz. Formal theories of action. In *The Frame Problem In Artificial Intelligence ed. Brown, F.*, pages 35–58. Morgan Kufmann Inc., 1987.

Lif87b. V. Lifschitz. On the declarative semantics of logic programs with negation. In *Foundations of Deductive Databases and Logic Programming ed. Minker, J.* Morgan Kaufmann Publishers Inc., 1987.

Lif88. V. Lifschitz. Pointwise circumscription. In *Readings in Non-Monotonic Reasoning*, pages 179–193. Morgan Kaufmann Publishers Inc., 1988.

Lif89. V. Lifschitz. Benchmark problems for formal non-monotonic reasoning, version 2.00. In *Lecture Notes In Artificial Intelligence 346*, pages 202–218. Springer-Verlag, 1989.

Lif91. V. Lifschitz. Circumscription. In: An Incomplete Draft of a chapter on Circumscription he sent me, 1991.

Llo84. J. W. Lloyd. *Foundations Of Logic Programming*. Springer-Verlag, 1984.

Llo87. J. W. Lloyd. *Foundations Of Logic Programming*. Springer-Verlag, second, extended edition edition, 1987.

Lov69. D. W. Loveland. Theorem-provers combining model elimination and resolution. In *Machire Intelligence 4 ed. Meltzer, B and Michie, D.*, pages 73–86. Edinburgh University Press, 1969.

Lov70. D. W. Loveland. A linear format for resolution. In *IRIA symp. on Automatic Demonstration*, pages 147–162. Springer-Verlag Berlin, 1970.

LS89. G. F. Luger and W. A. Stubblefield. *Artificial Intelligence and The Design of Expert Systems*. The Bengermin/Cummings Publishing Company Inc., 1989.

Luc70. D. Luckham. Refinements in resolution theory. In *IRIA symp. on Automatic Demonstration*, pages 163–190. Springer-Verlag Berlin, 1970.

Luk90. W. Lukaszewicz. *Non-monotonic Reasoning Formalization of Common Sense Reasoning*. Ellis Horwood Ltd., 1990.

Lüt88. S. Lüttringhaus. An interpreter with lazy evaluation for prolog with functions. In *Lecture Notes In Computer Science 385, CSL 88*, pages 199–217. Springer-Verlag, 1988.

LY91. L. Li and J. H. You. Making default inferences from logic programs. *Computational Intelligence*, 7:142–153, 1991.

MB88. R. Manthey and F. Bry. Satchmo: A theorem prover implemented in prolog. In *9th International Conference On Automated Deduction*, pages 415–434. CADE, 1988.

McC80. J. McCarthy. Circumscription- a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

McC86. J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

McD82. D. McDermott. Non monotonic logic ii ed. non monotonic modal theories. *Journal of the ACM*, 29(1):33–57, 1982.

McD91. D. McDermott. A general framework for reason maintenance. *Artificial Intelligence*, 50:289–329, 1991.

MD80. D. McDermott and J. Doyle. Non-monotonic logic 1. *Artificial Intelligence*, 13:41–72, 1980.

Mel66. B. Meltzer. Theorem-proving for computers: some results on resolution and renaming. *Computer Journal*, 8:341–343, 1966.

Min78. J. Minker. An experimental relational data base system based on logic or clause encounters of a logical kind. In *Logic and Data Bases ed. Gallaire, H. and Minker, J.*, pages 107–147. Plenum, New York, 1978.

Min82. J. Minker. On indefinite data bases and the closed world assumption. In *Lecture Notes in Computer Science 137*, pages 292–308. Springer-Verlag, 1982.

MMT88. M.A. McRobbie, R.K. Mayer, and P. B. Thistlewaite. Towards efficient 'knowledge-based' automated theorem proving for non-standard logics. In *9th International Conference On Automated Deduction*, pages 197–217. CADE, 1988.

Moo84. R. C. Moore. Possible-world semantics for autoepistemic logic. In *Proc. 1984 Non-Monotonic Reasoning Workshop*, pages 344–354. AAAI, 1984.

Moo85. R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–91, 1985.

Mor87. J. Morrissey. *A Treatement of Imprecise Data and Uncertainty in Information Systems*. PhD thesis, University College Dublin, 1987.

Mot87. P. L. Mott. A theorem on the consistency of circumscription. *Artificial Intelligence*, 31:87–98, 1987.

MP84. J. Minker and D. Perlis. Applications of protected circumcription. In *Lecture Notes In Computer Science 170*, pages 414–425. Springer-Verlag, 1984.

MP93. J. M. Morrissey and Viren Parasram. An approach to commonsense reasoning. In *Proceedings of the 6th Irish Conference on Artificial Science and Cognitive Science*, pages 197–200. Belfast University, 1993.

MR88. J. Minker and A. Rajasekar. Procedural interpretation of non-horn logic programs. In *9th International Conference On Automated Deduction*, pages 278–293. CADE, 1988.

MR89. H. Manilla and K.-J. Raïhä. Automatic generation of test data for relational queries. *Journal of Computer and System Sciences*, 38:240–258, 1989.

Nab91. Barkakati Nabajyoti. *X Window System Programming*. SAMS, 1991.

Nau89. J. F. Naughton. Data independent recursion in deductive databases. *Journal of Computer and System Sciences*, 38:259–289, 1989.

Nie88. I. Niemalä. Decision procedure for autoepistemic logic. In *9th International Conference On Automated Deduction*, pages 675–684. CADE, 1988.

Nie90. X.and Plaisted D. A. Nie. Refinements to depth-first iterative-deepening search in automatic theorem proving. *Artificial Intelligence*, 41:223–

235, 1989/90.

NM90. G. Nadathur and D. Miller. Higher-order horn clauses. *Journal of the ACM*, 37(4):807–814, 1990.

Nut83. J. T. Nutter. What else is wrong with non-monotonic logics? In *Proc. Fifth Conference of the Cognitive Science Society*, pages 1–5, 1983.

Nut87. J. T. Nutter. Uncertainty and probability. In *Proc. Tenth IJCAI Milan 1987*, pages 373–379, 1987.

NY78. J. M. Nicolas and K. Yazdanian. Integrity checking in deductive data bases. In *Logic and Data Bases ed. Gallaire, H. and Minker, J.*, pages 325–344. Plenum, New York, 1978.

Par94. Viren Parasram. An approach to commonsense reasoning : Default logic augmented with certainty factors. Master's thesis, University of Windsor, 1994.

Pea90a. J. Pearl. Chapter 6: Introduction. In *In Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 339–344. Morgan Kaufmann Publishers, Inc., California, 1990.

Pea90b. J. Pearl. Probabilistic semantics for nonmonotonic reasoning: A survey. In *In Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 699–710. Morgan Kaufmann Publishers, Inc., California, 1990.

Per86. D. Perlis. On the consistency of commonsense reasoning. *Computational Intelligence*, 2:180–190, 1986.

Per87. D. Perlis. Circumscribing with sets. *Artificial Intelligence*, 31:201–211, 1987.

Per88. D. Perlis. Autocircumscription. *Artificial Intelligence*, 36:223–236, 1988.

PM86. D. Perlis and J. Minker. Completeness results for circumscription. *Artificial Intelligence*, 28:29–42, 1986.

Poo88. D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

Poo91. D. Poole. The effect of knowledge on belief: Conditioning, specificity and the lottery paradox in default reasoning. *Artificial Intelligence*, 49:281–307, 1991.

Poo92. D. Poole. Local users guide to theorist. Technical report, University of British Columbia, 1992.

Prz89. T. C. Przymusinski. An algorithm to compute circumscription. *Artificial Intelligence*, 38:49–73, 1989.

Prz91. T. Przymusinski. Three-valued nonmonotonic formalisms and semantics of logic programs. *Artificial Intelligence*, 49:309–343, 1991.

Qui89a. *Quintus Prolog*, release 2.5 edition, 1989.

Qui89b. *Quintus ProWINDOWS*, release 1.1 edition, 1989.

RC81. R. Reiter and G. Criscuolo. On interacting defaults. In *Proc. of the Seventh International Join Conference On Artificial Intelligence*, pages 270–276. IJCAI, 1981.

Red86. U. S. Reddy. On the relationship between logic and functional languages. In *Logic Programming Functions, Relations and Equations ed. DeGroot,* *D. and Lindstrom, G.*, pages 3–36. Prentice-Hall, 1986.

Rei78. R. Reiter. On closed world data bases. In *Logic and Data Bases ed. Gallaire, H. and Minker, J.*, pages 55–76. Plenum, New York, 1978.

Rei80. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

Rei87. R. Reiter. Nonmonotonic reasoning. *Annual Review Computer Science*, 2:147–186, 1987.

Ric89. T. Richards. *Clausal Form Logic An Introduction to the Logic of Computer Reasoning*. Addison-Wesley Publishing Company, 1989.

Rob65. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

Rob68. J. A. Robinson. The generalised resolution principle. In *Machine Intelligence 3 ed. Michie, d.*, pages 77–93. Edinburgh University Press, 1968.

Ros89. Peter Ross. *Advanced Prolog Techniques and Examples*. Addison - Wesley Publishing Company, 1989.

Sak88. H. Sakai. Inference methods and semantics on or-type knowledge bases. In *Lecture Notes In Artificial Intelligence 383, Logic Programming 88*, pages 136–155. Springer-Verlag, 1988.

SB75. E.H. Shortliffe and B.G. Buchanan. A model for inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379, 1975.

SBV90. Danny de Schreye, M. Bruynooghe, and K. Verschaetse. On the existence of nonterminating queries for a restricted class of prolog-clauses. *Artificial Intelligence*, 41:237–248, 1989/90.

Sha76. G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey, 1976.

Sha89. E. Shapiro. The family of concurrent logic programming languages. *ACM Computing Surveys*, 21(3):412–512, 1989.

Sha90a. G. Shafer. Chapter 2: Introduction. In *In Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 7–14. Morgan Kaufmann Publishers, Inc., California, 1990.

Sha90b. G. Shafer. Chapter 7: Introduction. In *In Book: Readings in Uncertain Reasoning ed. Shafer, G. and Pearl, J.*, pages 473–481. Morgan Kaufmann Publishers, Inc., California, 1990.

Shi88. T. Shintani. An approach to non monotonic inference mechanism in production system kore/ie. In *Proc. of the 7th Conference on Logic Programming,Tokyo*, pages 38–52, 1988.

Sho76. E.H. Shortliffe. Mycin: Computer based medical consultation. *New York: American Elsevier*, 1976.

Sho86. Y. Shoham. Chronological ignorance: An experiment in nonmonotonic temporal reasoning. In *Proc. of the Fifth National Conference On Artificial Intelligence*, pages 389–393. AAAI, 1986.

Sho87. Y. Shoham. Nonmonotonic logics: meaning and utility. In *Proc. of Intl. Joint Conf. on AI*, pages 388–393. IJCAI, 1987.

SL88. G. Saake and U. W. Lipeck. Using finite-linear temporal logic for specifying data base dynamics. In *Lecture Notes In Computer Science 385,CSL 88*, pages 288–300. Springer-Verlag, 1988.

Sla67. J.R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697, 1967.

SMDP88. P. Smets, E.H. Mamdani, D. Dubois, and H. Prade. *Non-Standard Logics For Automated Reasoning*. Academic Press, 1988.

Sta89. R.F. Stark. A direct proof for the completeness of sld resolution. In *Lecture Notes In Computer Science,CSL 89*, pages 382–395. Springer-Verlag, 1989.

Str89. Martin Strobel. Non-monotonic reasoning and its applications (a survey). Technical report, University of Windsor Technical Report 89- 011, 1989.

SWC71. G. J. Sussman, T. Winograd, and E. Charniak. Micro-planner reference manual. *Technical Report 203A MIT AI Lab. and Cambridge MA*, 1971.

TF85. A. Takeuchi and K. Furukawa. Partial evaluation of prolog programs and its application to meta-programming. Technical Report TR-126, Institute for New Generation Computer Technology (ICOT), 1985.

Tou84. D. S. Touretzky. Implicit ordering of defaults in inheritance systems. In *Proc. of the Fifth National Conference On Artificial Intelligence*, pages 322–325. AAAI, 1984.

Ull89. J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1989.

Var86. M.Y. Vardi. Querying logical databases. *Journal of Computer and System Sciences*, 33:142–160, 1986.

Vol89. H. Volga. The semantics of disjunctive deductive data bases. In *Lecture*

Notes In Computer Science,CSL 89, pages 402–421. Springer-Verlag, 1989.

Wo90. K. Waugh and others. Designing squirrel: an extended sql for a deductive database system. *The Computer Journal*, 33(6):535–546, 1990.

WP88. T. Wakayama and T. H. Payne. Case inference in resolution based languages. In *9th International Conference On Automated Deduction*, pages 313–322. CADE, 1988.

WRC65. L. Wos, G.A. Robinson, and D.F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12(4):536–541, 1965.

YA88. H. Yuasa and S. Arikawa. Pseudo extensions in default reasoning and belief inference. In *Proc. of the 7th Conference on Logic Programming,Tokyo*, pages 27–37, 1988.

Zad65. L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

Zad78. L. Zadeh. Fuzzy sets as a basis for a theory of possibility. In *Fuzzy Sets and Systems*. North-Holland, Amsterdam, 1978.

Zad83. L. Zadeh. Commonsense knowledge representation based on fuzzy logic. *Computer*, 16(10):61–65, 1983.

Zet89. F. Zetzche. Non-monotonic reasoning with the atms. In *Proc. 4th Portugese Conference on Artificial Intelligence*, pages 119–128. EPIA, 1989.

ZM89. A. Zhang and W. Marek. On the classification and existence of structures in default logic. In *Proc. 4th Portugese Conference on Artificial*

Intelligence, pages 129–140. EPIA, 1989.

APPENDIX E
SOURCE CODE[‖] FOR WINIT

```
%#######################################################################%
%         ************************************
%         *             WINIT                *
%         *   Windsor Intellingent Thinker   *
%         * A Quintus Prolog Implementation *
%         ************************************

%         *************************************
%         * Programmer : VIREN PARASRAM       *
%         * Email : viren@uwindsor.ca         *
%         * ***********************************

% WINIT
% =====
% Winit is a Quintus Prolog implementation of CFDL.
% CFDL [Par94] is a Certainty Factor Default Logic.
% The documentation is divided into three segments:
%  I. Representation of CFDL in Prolog
%  II. WINIT ProWindow interface source code
%  III. WINIT source code

%#######################################################################%
% I. Representation of CFDL in Prolog
%    ================================
% Constants:- any Prolog constant
% Variables:- any Prolog constant
% Predicates:- any Prolog predicate
```

‖ I would like to thank David Poole for emailing me his Theorist *[Poo92]* source code. I have included his Prolog code for finding the most general unifier with the occurs check into my implementation.

```
% Term:- a constant or a variable
% Clause:- In CFDL q<-a1,a2,...,an : cf
%          In WINIT |cf|:sign(cf):q<- a1,a2,...,an

%  Example1:- In CFDL flies(X)<-bird(X),has_wings(X)  : 0.95
%             In WINIT 0.95:(+):flies(X)<-bird(X),has_wings(X)

%  Example2:- In CFDL flies(X)<-penguin(X)  :  -0.98
%             In WINIT 0.98:(-):flies(X)<-penguin(X)

%  Example3:- In CFDL teacher(joan)<-:1
%             In WINIT 1.0:(+):teacher(joan)

% Note in example 3 when there are no antecedents the implication
%   sign is dropped in WINIT.

% Note also that WINIT uses the prolog command
%  unix(system(Command)).
% to execute UNIX commands. These commands are used for:
% on line help, load and save KB

%####################################################################%
%  II. WINIT ProWindow interface source code
%      ===================================

%            **************************
%          *    WINIT Theorem Prover   *
%         *   ProWindows Interface      *
%        *  Programmer: Viren Parasram   *
%       ***********************************

% Note in the documentation to this section if a predicate is defined
% in section III a comment "Defined in non-ProWindows code" is added.
% Print_view loaded to allow use of:
%  print_view/2 to send output to a specified view.
:- ensure_loaded(library(print_view)).

% Popup_prompt loaded to use prompt/5 for user input.
:- ensure_loaded(library(popup_prompt)).

% Term_atom loaded to allow use of:
% atom_to_term/2 to convert prowindows atoms to prolog terms
:- ensure_loaded(library(term_atom)).

% Location of files
% *****************
% You will need to replace the path name
%  /home/ucc/perm/parasra/Thesis
% with the path name of the directory where you load
% WINIT's source code. Also the Help files must be placed
% in a Help subdirectory and the bitmap icons in the
% Icons subdirectory.
% The following file names are used by WINIT:
%  help_areas, saved_states, saved_kbs, winit, winit_kb

% Source Code for WINIT in file
%  /home/ucc/perm/parasra/Thesis/winit.pl
% Help files in Directory
%  /home/ucc/perm/parasra/Thesis/Help
% Saved states in Directory
%  /home/ucc/perm/parasra/Thesis/States
% Saved Knowledge Bases in Directory
%  /home/ucc/perm/parasra/Thesis/KB
% Bitmap Icons in Directory
%  /home/ucc/perm/parasra/Thesis/Icons
% setup_help/0
% ============
% setup_help copies the names of the files in the Help directory
% into the help_areas file which is used by the help browser.

setup_help :-
     name(Cmd,"ls /home/ucc/perm/parasra/Thesis/Help  >
help_areas"),
     unix(system(Cmd)).

% setup_states/0
% ==============
% Setup_states copies the names of the files in the State directory
% into the saved_states file which is used to remind the user of
% the names of the saves states.

setup_states :-
     name(Cmd,"ls /home/ucc/perm/parasra/Thesis/States  >
saved_states"),
     unix(system(Cmd)).

% setup_kbs/0
% ==============
% setup_kbs copies the names of the files in the KB directory
% into the saved_kbs file which is used by the load_kb browser

setup_kbs :-
     name(Cmd,"ls /home/ucc/perm/parasra/Thesis/KB  > saved_kbs"),
     unix(system(Cmd)).

% begin/0
% """""""
% begin is used to open the ProWindos interface to WINIT
```

```
begin :-
     asserta(state_names([])),
     setup_winit_windows,
     open_winit_window.

% setup_winit_windows/0
% ====================
% Creates the window objects.
% The window sizes may be changed depending on your monitor.
% This can be done using the mouse or changing the x-y size values.

setup_winit_windows :-
     new(@main, view('WINIT (Windsor Intelligent Thinker)')),
       send(@main, size, size(595, 420)),
       send(@main, editor, cascade(@main,commands,0)),
       send(@main, edit_mode, command),
     new(Menu_bar,dialog('WINIT (Windsor Intelligent Thinker)')),
     new(@main_bar,menu_bar(winit)),
     new(@help,popup('HELP',help)),
     new(@file,popup('FILE',file)),
     new(@query,popup('QUERY',query)),
     new(@kb,popup('KB',kb)),
     new(@change,popup('Change clause(s)',change)),
     new(@add,popup('Add clause(s)',add1)),
     new(@delete,popup('Delete clause(s)',delete1)),
       send(@main_bar,append,[@file,@kb,@query,@help]),
       send(@help,append,['Help','Version','hot Keys']),
       send(@query,append,query),
       send(@kb,append,['Load', 'Save',@change]),
       send(@file,append,['save state','Save kb']),
       send(@change,append,[@add,@delete]),
       send(@add,append,['add 1','add #']),
       send(@delete,append,['a clause','all clauses']),
        send(Menu_bar,append,@main_bar),
        send(Menu_bar,above,@main),
      new(@icon,bitmap(64,64,1)),
         load_icon(winit),
        send(@main,icon,@icon).

% load_icon/1
% ***********
% Loads the 64 bit X-bitmap file with name in arg 1
% from the Icons subdirectory into the @icon bitmap object.

load_icon(Icon_name) :-
        add_prefix("Icons/",Icon_name, Icon),
           send(@icon,load,Icon).

% open_winit_window/0
% *******************
% Opens the main WINIT window with its menu bar.

open_winit_window :-
      send(@main,open,point(2,70)).

% commands/2
% **********
% commands is used to give the hot key definition.
% The main WINIT window can be in command or insert mode.
% In command mode entering a single character will cause the
% actions defined below. Note all these actions can also be
% triggered using the mouse and menu bar.

commands(@main, a) :- add1.
commands(@main, h) :- make_help.
commands(@main, k) :- hot_keys_msg.
commands(@main, v) :- version_msg.
commands(@main, escape) :-
     send(@main, edit_mode, command),
     print_view(@main, 'In command mode'),
     send(@main, newline).
commands(@main, i) :-
     send(@main, edit_mode, insert),
     print_view(@main, 'In insert mode. Use control_a instead of
Enter.'),
     send(@main, newline),
     print_view(@main, ' Bug in ProWINDOWS 2'),
     send(@main, newline).

% This next option processes user input when in the input mode
% "newline" should have been used instead of "control_a" but there
% is a bug in ProWindods since the @main view's message is not
% executed when Return or Line Feed is pressed.
% This bug is not present in newer versions of ProWindows.
% Hence in insert mode I simply process the first key as a
% Hot key in the command mode as defined above.

commands(@main, control_a) :-
     get(@main,line, string(Cmd)),
     send(@main, newline),
     print_view(@main, 'Processing your input: '),
     print_view(@main, Cmd),
     send(@main, newline),
     name(Cmd, [Key|Rest]),
     name(Hot_key, [Key]),
     commands(@main, Hot_key).

% Trap for invalid command key
commands(View,Key) :-
```

```
        print_view(View,'Command key "'),
        print_view(View,Key),
        print_view(View,'"not recognised.'),
        send(@main,newline).

%==================================================================%

% help/2
% ******
% Executes coresponding predicate when help popup selected.

help(@help, 'Help') :- make_help.
help(@help, 'Version') :- version_msg.
help(@help, 'hot Keys') :- hot_keys_msg.

%------------------------------------------------------------------%
% Open help help windows if they exist.
make_help :-
        object(@help_view),
        print_view(@main,'Help window being opened.'),
        send(@main,newline),
        open_help_windows.

% Create and open Help Windows
make_help :-
        create_help_windows,
        open_help_windows.

open_help_windows :-
        send(@help_view,open,point(500,80)),
          setup_help, % used to get the most recent list of help files
        send(@help_browser,load,help_areas),
        send(@help_browser,sort).
```

```
create_help_windows :-
     new(@help_browser,browser('HELP')),
     new(@help_view,view('HELP')),
     print_view(@main,'Help Activated'),
     send(@main,newline),
     send(@help_view,right,@help_browser),
     send(@help_view,size,size(350,300)),
       load_icon(help),
     send(@help_view,icon,@icon),
     send(@help_browser,selected,cascade(@help_view,get_help,0)).

% get_help/2
% **********
% Loads the help file selected into the view window.
get_help(@help_view,X):-
  add_prefix("Help/",X,Path),
  send(@help_view,load,Path).

%-------------------------------------------------------------%

version_msg :-
     print_view(@main,'This is WINIT version 1.0'),
     send(@main,newline).

%-------------------------------------------------------------%
% Open hot key window if it exists
hot_keys_msg :-
     object(@hot_keys),
     print_view(@main,'Reopening window with command keys definitions'),
     send(@main,newline),
     open_hot_key_windows.

% Creates and opens hot key window
```

```
hot_keys_msg :-
     print_view(@main,'View window has command keys definitions'),
     send(@main,newline),
     create_hot_key_windows,
     open_hot_key_windows.

create_hot_key_windows :-
     new(@hot_keys,view('WINIT HOT KEYS')),
     send(@hot_keys,size,size(400,300)),
       load_icon(hot_keys),
     send(@hot_keys,icon,@icon).

open_hot_key_windows :-
     send(@hot_keys,open,point(490, 90)),
     add_prefix("Help/",hot_keys,Path),
     send(@hot_keys,load,Path). % Load most recent hot key file
%==================================================================%
% file/2
% ******
% Executes coresponding predicate when file popup selected.

file(@file, 'save state') :- make_save_state.
file(@file,'Save kb') :- save_kb.

%------------------------------------------------------------------%
% make_save_state/0
% *****************
% Pops up windows to help save the present execution state of the
% program under a user specified name or as winit by default.

:- dynamic saved_state/1.
:- dynamic state_names/1.
```

```
make_save_state :-
     print_view(@main,'Save state Activated'),
     send(@main,newline),
        new(Dialog, dialog('save state')),
        new(Label, label('Save present state of program')),
        send(Dialog, append, Label),
        new(Save, button('Save', pressed_save)),
     new(Saveas,button('Save as', pressed_save)),
        new(List_names,button('list names', pressed_save)),
        new(Cancel, button('Cancel', pressed_save)),
     send(Label, above, Save),
        send(Save, left, Saveas),
     send(Saveas, left, List_names),
     send(List_names, left, Cancel),
        send(Dialog, open).

pressed_save(Dialog, 'Save') :-
     saved_state(State), % Use most recent saved state name
     save_state1(State) ,
     send(Dialog, destroy).

pressed_save(Dialog, 'Save') :-
     asserta(saved_state(winit)), % Use default save state name
     update_state_names(winit),
     save_state1(winit),
     send(Dialog, destroy).

pressed_save(Dialog, 'Save as') :-
     new(File_name_box,dialog('Enter file name')),
     new(Name, text_item('File
Name','',cascade(File_name_box,dummy,0))),
     send(File_name_box,append,Name),
     send(File_name_box,open),
    get(@sys,wait,message(File_name_box,dummy,Input)),
    abolish(saved_state,1),
    asserta(saved_state(Input)),
    save_state1(Input),
    update_state_names(Input),
    send(File_name_box,destroy),
    send(Dialog, destroy).

pressed_save(Dialog, 'list names') :-
        print_view(@main,'Old saved states names being listed.'),
      send(@main,newline),
    new(@state_names,view('WINIT SAVED STATES NAMES')),
      send(@state_names,size,size(400,300)),
      send(@state_names,open,point(500,90)),
        load_icon(state_names),
      send(@state_names,icon,@icon),
        state_names(Names),
      send_list_into_view(Names,@state_names),
      send(Dialog, destroy).

pressed_save(Dialog, 'Cancel') :-
    print_view(@main, '  Save State cancelled'),
    send(@main,newline),
    send(Dialog, destroy).

update_state_names(Name) :-
    state_names(List_names),
    (member(Name,list_names) ->
        True|
        (append([Name],List_names,New_list_names),
          retract(state_names(_)),
          asserta(state_names(New_list_names)))).
```

```
% SAVE_STATE1/1
% *************
% Saves the state of the program in the file name specified in arg 1.
save_state1(File_name) :- save_program(File_name,program_id),
     print_view(@main,'     System state saved in the file: '),
     print_view(@main,File_name),
     send(@main,newline),
     print_view(@main,'    To restart, from UNIX prompt type: '),
     print_view(@main,File_name),
     send(@main,newline).


% send_list_into_view1/2
% **********************
% Prints each member of a list separated by a blank space into a view.
send_list_into_view1([],View) :- send(View,newline).
send_list_into_view1([First|Rest], View) :-
     print_view(View,First),
     print_view(View,' '),
     send_list_into_view1(Rest,View).

% send_list_into_view1/2
% **********************
% Prints each member of a list separated by a newline into a view.
send_list_into_view([],View) :- send(View,newline).
send_list_into_view([First|Rest], View) :-
     print_view(View,First),
     send(View,newline),
     send_list_into_view(Rest,View).

%==================================================================%
% kb/2
% ****
% Executes coresponding predicate when @kb popup selected.

kb(@kb,'Load') :- load_kb.
kb(@kb,'Save') :- save_kb.

%-------------------------------------------------------------%
% load_kb/1
% *********
% Pops up windows to help load a KB previously saved.

load_kb :-
     print_view(@main,'Load KB Activated'),
     send(@main,newline),
     new(@load_browser,browser('LOAD KB')),
     new(@load_view,view('LOAD KB')),
       send(@load_view,right,@load_browser),
       send(@load_view,size,size(350,300)),
        send(@load_view,open,point(480,90)),
       load_icon(load_kb),
     send(@load_view,icon,@icon),
      setup_kbs,
       send(@load_browser,load,saved_kbs),
       send(@load_browser,sort),
       send(@load_browser,selected,cascade(@load_view,get_load,0)).

get_load(@load_view,X) :-
  add_prefix("KB/",X,Path),
  send(@load_view,load,Path),
  prompt('About to load the
file','Name',X,['OK','Cancel'],Name,Button,point(500,320)),
  load1(Name,Button).

load1(Name, 'OK') :-
```

```
     current_input(Oldinput),
     add_prefix("/home/ucc/perm/parasra/Thesis/KB/",Name,File),
      open(File,read,Input),
     set_input(Input),
     read(T),
     read_all(T),
     set_input(Oldinput),
     print_view(@main,File),
     send(@main,newline),
     print_view(@main,'  has been added to Knowledge Base.'),
     send(@main,newline),
     send(@load_browser, destroy).

load1(Name, 'Cancel') :- !.

% read_all/1
% **********
% Reads and asserts data from a file until the end of the file.

read_all(end_of_file) :- !.
read_all(T) :-
     assertz(T),
     read(Next),
     read_all(Next).

%==============================================================%
% save_kb/0
% *********
% Pops up windows to help save all the clauses in the current
% KB of the form A:B into a file specified or winit_kb by default.

:- dynamic saved_kb/1.
```

```
save_kb :-
     print_view(@main,'Save KB Activated'),
        send(@main,newline),
     new(Dialog, dialog('SAVE KB')),
     new(Label, label('Save all rules and facts in KB')),
        send(Dialog, append, Label),
     new(Save, button('Save', pressed_kb)),
     new(Saveas,button('Save as', pressed_kb)),
     new(Cancel, button('Cancel', pressed_kb)),
        send(Label, above, Save),
        send(Save, left, Saveas),
        send(Saveas, left, Cancel),
        send(Dialog, open,point(400,100)).

pressed_kb(Dialog, 'Save') :-
     saved_kb(State),
     abolish(saved_kb,1),
     asserta(saved_kb(State)),
     save_kb(State) ,
     print_view(@main,'  KB saved in file: '),
     print_view(@main,State),
     send(@main,newline),
     send(Dialog, destroy).

pressed_kb(Dialog, 'Save') :-
     asserta(saved_kb(winit_kb)),
     save_kb(winit_kb),
     print_view(@main,'  KB saved in file: winit_kb'),
     send(@main,newline),
     send(Dialog, destroy).

pressed_kb(Dialog, 'Save as') :-
     new(File_name_box,dialog('Enter file name')),
     new(Name, text_item('File
Name','',cascade(File_name_box,dummy,0))),
        send(File_name_box,append,Name),
        send(File_name_box,open),
        repeat,
        get(@sys,wait,message(File_name_box,dummy,Input)),
        abolish(saved_kb,1),
        asserta(saved_kb(Input)),
        save_kb(Input),
        print_view(@main,'  KB saved in file: '),
        print_view(@main,Input),
        send(@main,newline),
        send(File_name_box,destroy),
        send(Dialog, destroy).

pressed_kb(Dialog, 'Cancel') :-
     print_view(@main, '  Save KB cancelled'),
     send(@main,newline),
     send(Dialog, destroy).

%=================================================================%

% add1/2
% ******
% Pops up windows which direct you to add a clause
% or several clauses as selected.

add1(@add,'add 1') :-
     new(Dialog, dialog('Go to Prolog Window')),
     new(Label, label('Enter clause in prolog window')),
       send(Dialog, append, Label),
     new(Button, button('OK', prompt_button)),
       send(Button, below, Label),
        send(Dialog,open),
        repeat,
           get(@sys, wait, message(Dialog, prompt_button, 'OK')),
        send(Dialog, destroy),
           add, % Defined in non-ProWindows code
      print_view(@main,'Control returned to ProWINDOWS'),
         send(@main,newline).

add(@add,'add #') :-
     print_view(@main,'Add clauses Activated.'),
     send(@main,newline),
     print_view(@main,'   Enter number of clauses to be added, in
Prompter then,'),
          send(@main,newline),
     print_view(@main,'      Enter clauses, in Prolog window.'),
     send(@main,newline),
   prompt('  How many clauses to update
?','Number',1,['OK','Cancel'],Name,Button,point(520,320)),
       atom_to_term(Name,Num),
       update1(Num,Button),
        print_view(@main,'Control returned to ProWINDOWS'),
       send(@main,newline).

update1(Num, 'OK') :-
     add(Num).  % Defined in non-ProWindows code

update1(_, 'Cancel') :-
     print_view(@main,'Update KB cancelled at your request'),
     send(@main,newline).

%-----------------------------------------------------------%

delete1(@delete,'a clause') :-
```

```
     prompt('  Enter clause to be deleted',
'Clause','',['OK','Cancel'],Name,Button,point(520,320)),
  atom_to_term(Name,Num),
  delete2(Num,Button),
       print_view(@main,'  Control returned to ProWINDOWS.'),
      send(@main,newline).

delete1(@delete,'all clauses') :-
          prompt('  Warning this option deletes all
clauses','','',['OK','Cancel'],_,Button,point(520,320)),
      delete_all(Button).

delete_all('OK') :-
      clean, % Defined in non-ProWindows code
      print_view(@main,'All clauses removed from KB.'),
      send(@main,newline).

delete_all('Cancel') :-
      print_view(@main,'Clause deletion cancelled.'),
      send(@main,newline).


query(@query,query) :-
      new(Dialog, dialog('Go to Prolog Window')),
      new(Label, label('Enter queries at WINIT prompt
 in prolog window')),
      send(Dialog, append, Label),
      new(Button, button('OK', prompt_button)),
      send(Button, below, Label),
      send(Dialog,open),
      repeat,
           get(@sys, wait, message(Dialog, prompt_button, 'OK')),
           winit,
     print_view(@main,'Control returned to ProWINDOWS'),
          send(@main,newline),
     send(Dialog, destroy).

delete2(Num, 'OK') :-
          deletepw(Num).

delete2(_, 'Cancel') :-
     print_view(@main,'Update KB cancelled at your request'),
     send(@main,newline).

% Delete a clause if it exists.
deletepw(Cf-Clause) :- !,
  find_clause(Cf-Clause,Cf1-Sign),
 send_list_into_view1(['Clause'' ',Clause,' ',Sign,Cf1,'''
   about to be removed y/n: '],@main),
  (get0(121) -> (retract(Cf1-Sign-Clause),
      send_list_into_view1(['Clause'' ',Clause,' ',Sign,Cf1,'''
   has been deleted.'],@main),
          send(@main,newline))|
     (print_view(@main,' Clause NOT deleted.'),
          send(@main,newline))).
deletepw(Clause) :-
     deletepw(_-Clause).



% Window Utility Predicates
% *************************

% Dummy/2
% *******
% Dummy is used to facilitate use of cascade in a dialog box to get
```

```
input
dummy(_,_).



% Programming Utility Predicate
% """"""""""""""""""""""""""""
% These are predicates I found usefull when writing and debuging
the program.

% You can include any objects in the list which you wish to destroy
free :- free_win([@main,@main_bar,@help,@file,@query,@kb,@change]).

free_win([]).
free_win([Win|Rest]):- object(Win),send(Win,destroy),free_win(Rest).
free_win([Win|Rest]):- free_win(Rest).

% recompile.
c :-['winit.pl'].


%#######################################################################%

%        ***************************************
%        *       WINIT Theorem Prover          *
%        *  Source Code Common to Window &     *
%        *       Non-Window Versions           *
%        *  Programmer: Viren Parasram         *
%        ***************************************

:- op(1150,fy,h).       % Used to facilitate online help
:- op(1150,fx,set).     % Set turns flag on/off
:- op(1150,fx,flag).
:- op(1150,fx,save_kb).
:- op(1125,xfy,'<-').   % For representation of clauses
:- dynamic choice_pts/1.
:- dynamic goal/1.
:- dynamic active_list/1.
:- dynamic resolvant/1.
:- dynamic all_answers/1.
:- dynamic answers/1.

:-compile(library(basics)). /* for member/2 */

% WINIT/0
% *******
% This predicate invokes the WINIT interpreter.
% The control is put to the theorem prover and remains there until
% "quit." is entered at the "WINIT" prompt.

winit :-
     initialize_flags,
     unix(system(clear)),
     theorem_prover(winit,start).

initialize_flags :-
     killflags,
     asserta((flag (goal_variable_free,off))),
     asserta((flag (newclause,on))),
     asserta((flag (occurs_check,off))),
     asserta((flag (calculus_type,1))),
     asserta((flag (trace,off))).

set Flag,Val :- retract((flag (Flag,_))),
          asserta((flag (Flag,Val))).
set Flag,_ :- format('~nFlag "~w" does not exist.',[Flag]).
```

```
flags :- listing(flag).

% THEOREM_PROVER/2
% ****************
% This predicate reads a goal and executes it based on the "xecute"

% predicate's rule. This loops until the quit goal is entered.

theorem_prover(_,Status) :- Status == quit.
theorem_prover(_,_) :- format('~nWINIT : ',[]),
     read(Goal),!,
     xecute(Goal,Status),
     theorem_prover(_,Status).

% XECUTE/2
% *********
% This predicate gives the rules on how each goal should be
executed.

% First check if the boundary condition is satisfied.
xecute(Goal,Status) :-
  member(Goal,[quit,q,stop,end,end_of_file]),
  Status = quit.

% Next if goal user defined then execute.
xecute(Goal,_) :-

member(Goal,[save_state,add,add(_),delete(_),update(_),h,(h_),c,l
,flags,set(_),clean,add_db(_)]),
  Goal.

% Answer Users Query
xecute((Q1,Q2),_):-
```

```
  xecute(Q1,_),xecute(Q2,_).

xecute(Query,_):-
  check(Query),
  solve(Query,Ans).

check(Query) :-
  variable_free(Query),
  (set (goal_variable_free,on)).
check(_) :-
  (set (goal_variable_free,off)).


% Resolution Strategy
solve(Goal,Ans) :-
  initialize,
  retract(goal(_)),asserta(goal((1.0-'+')-Goal)),
  loop_block.

loop_block :-
 show(loop_block),
  choice_pts(Choice_pts),goal(Cf_goal-Goal),
  (pop(Goal,Choice_pts,New_choice_pts,Goals_c_ls) ->
    (retract(choice_pts(_)),
     asserta(choice_pts(New_choice_pts)),
     (set newclause,off),
     resolve(Goal,Goals_c_ls,C2_n),sub_block(C2_n))|
    (search(Goal,List),
     (List = [] -> (backtrack)|
                  ((set newclause,on),
              resolve(Goal,List,C2_n),sub_block(C2_n))))).

 sub_block(C2_n) :-
```

```
  test_C_list(C2_n),
  test_resolvant,
  test_active_ls.

backtrack :- show(backtrack),test_active_ls.

test_active_ls :-
  active_list(Active_ls),
  (Active_ls = [] -> check_answer_ls |
             (update_goal(Active_ls),
              test_ans_ls)).

check_answer_ls :-
  answers(Ans),
  Ans = [] -> (write(' NO by CWA'))|
           report.

test_ans_ls :- answers(Ans_ls),
  (Ans_ls = [] -> (pop_active_list_loop)|
             (compare_cf_ans_goal)).

compare_cf_ans_goal :-
  goal((Cf_goal-_)-Goal),
  answers([(Cf_ans-_)-Ans|T]),
  (Cf_goal @< Cf_ans ->
     report|
     pop_active_list_loop).
pop_active_list_loop :-
     retract(active_list([H|T])),
     asserta(active_list(T)),
     loop_block.
```

```
pop_active_list :- retract(answers([H|T])),
            asserta(answers(T)).

update_goal([H|T]) :-
  retract(goal(_)),
  asserta(goal(H)).

test_resolvant :-
  resolvant(((Cf_res-Res)-Leaf)),
  (Res = [] -> (Leaf =.. [_|Binding],test_binding(Cf_res-Binding))|
           (retract(active_list(Active_ls)),
         append([Cf_res-Res],Active_ls,New_active_ls),
         cf_sort(New_active_ls,New),
         asserta(active_list(New)))).

test_binding(Cf-Binding) :-
  retract(answers(Ans)),
  retract(all_answers(All_ans)),
  (member((_-Binding),Ans) ->
update_ans_bindings((Cf-Binding),Ans,New_ans) |
     (append([(Cf-Binding)],Ans,New_ans))),
  asserta(answers(New_ans)),
  append(New_ans,All_ans,New_all_ans),
  asserta(all_answers(New_all_ans)).

update_ans_bindings((Cf-Binding),[(Cf1-Binding)|Rest],[Better_ans
|Rest]):-
  Cf = Abs_cf-Sign, Cf1 = Abs_cf1-Sign1,
  (Abs_cf @> Abs_cf1 -> Better_ans = Cf | Better_ans = Cf1).
update_ans_bindings((Cf-Binding),[(Cf1-Old)|Rest],[(Cf1-Old)|Othe
r]) :-
  update_ans_bindings((Cf-Bindings),Rest,Other).
```

```
test_C_list([]).
test_C_list(C_list) :-
  (flag newclause,on),
  add_choice_pts(C_list,Goal,Cf_goal),
  generate_choices(Goal,Cf_goal,C_list,Choices),
       update_active_ls(Choices).
test_C_list(C_list) :-
  (flag newclause,off),
   add_choice_pts(C_list,Goal,Cf_goal).

add_choice_pts(C_list,Goal,Cf_goal) :-
  goal(Cf_goal-Goal),
  retract(choice_pts(Choice_pts)),
  append([Goal-C_list],Choice_pts,New_choice_pts),
  asserta((choice_pts(New_choice_pts))).

update_active_ls(Choices) :-
  retract(active_list(Active_ls)),
  append(Choices,Active_ls,New_active_ls),
  cf_sort(New_active_ls,New),
  asserta(active_list(New)).

generate_choices(G,Cf_goal,[],[]).
generate_choices(G,Cf_goal,[Cf1-_|T],[Cf-G|Rest]) :-
  new_cf(Cf_goal,Cf1,Cf),
  generate_choices(G,Cf_goal,T,Rest).

resolve(Goal,[Cf_cl-Clause|T],T) :-
  goal(Cf_goal-Goal),
  ((flag newclause,on) -> new_cf(Cf_goal,Cf_cl,Cf_resolvant)|
                  (Cf_resolvant = Cf_goal)),
  retract(resolvant(_)),
  (\+(Clause = (Goal <- Antecedents)) ->
     asserta(resolvant((Cf_resolvant-[])-Clause))|
     (Clause = (Goal <- Antecedents),
    asserta(resolvant((Cf_resolvant-Antecedents)-[])))).

%===================================================================%
% Calculus Module
% You can define different calculi here. The default is Calculus 1
% You can use any unused number for your calculus.

new_cf(C1,C2,Cf) :- (flag calculus_type,1), cf1(C1,C2,Cf).
% For a new calculus you would use a line like the one
%  commented out below
% new_cf(C1,C2,Cf) :- (flag calculus_type,2), cf2(C1,C2,Cf).
% Then add the appropriate cf2 predicate definitions.
% You will also need to set the calculus_type flag
%  to the desired calculus

% CALULUS 1
cf1(_-_,1.0-'-',0.0-'-').
cf1(Cf1-'+',Cf2-'+',Cf3-'+') :- Cf3 is Cf1 * Cf2.
cf1(Cf1-_,Cf2-_,Cf3-'-') :- Cf3 is Cf1 * Cf2.
%===================================================================%

search(Goal,List) :-
     findall(Clause,get_match(Goal,Clause),Clause_list),
     (Clause_list = [] -> List = []|
                 cf_sort(Clause_list,List)).

get_match(Goal,Clause) :-
     Y-X,(unif(X,(Goal<-R));unif(X,Goal)),Clause = Y-X.

% Initial lists for resolution
```

```
goal([]).
choice_pts([]).
active_list([]).
resolvant([]).
answers([]).
all_answers([]).



initialize :-  retract(goal(_)),
          retract(choice_pts(_)),
          retract(active_list(_)),
          retract(resolvant(_)),
          retract(answers(_)),
          retract(all_answers(_)),
          asserta(goal([])),
          asserta(choice_pts([])),
          asserta(active_list([])),
          asserta(resolvant([])),
          asserta(answers([])),
          asserta(all_answers([])).

% PROGRAM_ID/0
% ************
% This predicate is executed every time the program is
% restarted from a saved state.
% It displays welcome,version and help messages.

program_id :- format('~n
     ************************************
     *     Welcome to WINIT ver 1.0    *
     *     For help type ''h.''          *
     ************************************~n',[]).
```

```
% SAVE_STATE/0
% ************
% Saves the state of the program in the default filename 'winit'.

save_state :-save_state(winit).


% SAVE_STATE/1
% ************
% Saves the state of the program in the file name specified in arg 1.

save_state(File_name) :- save_program(File_name,program_id),
      format('~n
    System state saved in the file ''~w''.
    To restart, from UNIX prompt typE
''~w''.~n',[File_name,File_name]).


% SAVE_KB/1
% *********
% Saves the KB into the specified file

(save_kb Kb) :- !,
     add_prefix("/home/ucc/perm/parasra/Thesis/KB/",Kb,File_name),
     tell(File_name),
     ls,
     told.

%------------------------------------------------------------
-----------
% H/0
% ***
% Used to display help options.

 (h) :- format('~n
    This is WINIT version 1.0 (all comments to Viren Parasram).
        Help is available in the following areas:',[]),nl,
       unix(system('ls /tmp_mnt/home/ucc/perm/parasra/Thesis/Help
| more')),
       format('~n
    Enter your choice followed by <return>~n',[]).

% h H
% ***
% Displays help in a specific topic

(h H) :- !,
      add_prefix("more
/home/ucc/perm/parasra/Thesis/Help/",H,Cmd),
      unix(system(Cmd)).
%------------------------------------------------------------
-------------
%
% Predicates to input clauses with certainty factors.
%

% Add a clause by prompting user for input.
add:- get_clause(Clause),
      get_cf(Cf),
      test_cf(Cf,Clause).

test_cf(Cf,Clause) :- Cf @>
0.0,assert_if_not_present((Cf-'+')-Clause).
test_cf(Cf,Clause) :- Cf @< 0.0,
     Cf1 is Cf * -1.0,assert_if_not_present((Cf1-'-')-Clause).
test_cf(Cf,Clause) :- Cf =
0.0,assert_if_not_present((Cf-'-')-Clause).
```

```
assert_if_not_present((Cf-Sign)-Clause) :-
     call((Cf-Sign)-Clause),
     format('~nClause exists~n',[]).
assert_if_not_present((Cf-'+')-Clause) :-
     call((Cf-'-')-Clause),
     format('~nAddition will cause a direct conflict
Replace old clause ? (y/n)
',[]),get0(X),resolve_conflict(X,(Cf-'-')-Clause).
assert_if_not_present((Cf-'-')-Clause) :-
     call((Cf-'+')-Clause),
     format('~nAddition will cause a direct conflict
Replace old clause ? (y/n)
',[]),get0(X),resolve_conflict(X,(Cf-'+')-Clause).
assert_if_not_present((Cf-Sign)-Clause) :-
asserta((Cf-Sign)-Clause).

resolve_conflict(121,(Cf-'-')-Clause) :-
     retract((Cf-'-')-Clause), asserta((Cf-'+')-Clause).
resolve_conflict(121,(Cf-'+')-Clause) :-
     retract((Cf-'+')-Clause), asserta((Cf-'-')-Clause).
resolve_conflict(_,_) :-
     format('~nNo change to KB.',[]).

% Add n number of clauses
add(0).
add(X) :- add,
       Y is X - 1,
       add(Y).


% Update an existing clause.
update((Cf-Clause)) :- !,
          find_clause(Cf-Clause,Cf1-Sign),
          retract((Cf1-Sign)-Clause),
          format('~nOld Clause : ~w : ~w ~2f
.~n',[Clause,Sign,Cf1]),
             add.
update(Clause) :- update((_-Clause)).
update(_) :- format('~nNo such clause in the knowledge base',[]).

% Delete a clause if it exists.
delete(Cf-Clause) :- !,
  find_clause(Cf-Clause,Cf1-Sign),
  format('~nAbout to remove the clause:~n~5|~w : ~w ~2f . ~nRemove
y/n:',[Clause,Sign,Cf1]),
  (get0(121) -> (retract(Cf1-Sign-Clause),format('~n~w : ~w ~2f
deleted.~n',[Clause,Sign,Cf1]))|
     (format('Clause NOT deleted.~n',[]))).
delete(Clause) :-
     delete(_-Clause).


find_clause(Cf1-Clause,Cf-Sign) :-
     number(Cf1), abs(Cf1,Sign,Cf).
find_clause(Cf1-Clause,Cf) :- var(Cf1),!,(call(Cf-Clause)
->true|(format('~n~w not in KB~n',[Clause]),fail)).
find_clause(Cf1-Clause,Cf) :- fail.


abs(Number,Abs_num,Sign) :- Number @>= 0,Abs_num is Number * 1.0,
Sign = '+'.
abs(Number,Abs_num,Sign) :- Number @< 0,Abs_num is Number * -1.0,
Sign = '-'.

%
% Predicates to help input Clauses and Certainty factors
```

```
%
get_clause(Clause) :-
     write('New Clause : '),
     read(Clause).
get_cf(Cf) :-
     write('Certainty Factor : '),
     get_real(Cf).

get_real(Real_num) :-
     get_goal(Ascii_list),
     check_decimals(Ascii_list,New_ascii_list),
     number_chars(Num,New_ascii_list),          /* number_chars
built in f'cn */
     Real_num is Num * 1.0.

check_decimals(List,New_list):-
     insert_zero_before_leading_decimal(List,List1),
     remove_trailing_decimal(List1,New_list).

insert_zero_before_leading_decimal([46|Rest],[48,46|Rest]).
insert_zero_before_leading_decimal([45,46|Rest],[45,48,46|Rest]).
insert_zero_before_leading_decimal(List,List).

remove_trailing_decimal(List,Out_list):-
     match_last(List,46,Out_list).
remove_trailing_decimal(List,List).

match_last([Ele],Ele,[]).
match_last([X|Tail],Ele,[X|New_tail]) :-
match_last(Tail,Ele,New_tail).

% Process input.
get_goal(Goal) :- get0(Char),
            get_input(Char,Goal).

get_input(10,[]).
get_input(32,Others) :-
     get0(Next),get_input(Next,Others).
get_input(Char,[Char|Others]):-
     get0(Next),get_input(Next,Others).



%------------------------------------------------------------
%
% Test data
% The code to read clauses from a file is in the Prowindows section
% Thus these test data can be placed in files
%
birddata :-
     asserta((1.0-'+')-(bird(X) <- emu(X))),
     asserta((1.0-'+')-(bird(X) <- robin(X))),
     asserta((1.0-'+')-bird(tweety)),
     asserta((1.0-'-')-bird(john)),
     asserta((0.0-'+')-bird(alien)),
     asserta((0.8-'+')-(flies(X) <- bird(X))).
bird :-
     asserta((0.8-'+')-(flies(X) <- bird(X))),
     asserta((0.7-'+')-(flies(X) <- bat(X))),
     asserta((0.99-'-')-(flies(X) <- dead(X))),
     asserta((0.7-'+')-bird(ben)),
     asserta((0.9-'+')-bat(ben)),
     asserta((1.0-'+')-dead(tweety)),
     asserta((1.0-'+')-bird(tweety)).
best :-
     asserta((0.9-'+')-(a(X) <- b1(X))),
     asserta((0.8-'+')-(a(X) <- b2(X))),
     asserta((0.7-'+')-(a(X) <- b3(X))),
     asserta((0.9-'+')-(b2(X) <- c1(X))),
     asserta((1.0-'+')-c1(c1)),
     asserta((0.9-'+')-(b3(X) <- c2(X))),
     asserta((0.9-'+')-(c2(X) <- d1(X))),
     asserta((0.8-'+')-(c2(X) <- d2(X))),
     asserta((1.0-'+')-d1(d1)),
     asserta((1.0-'+')-b1(b1)),
     asserta((1.0-'+')-d2(d2)).

a1 :-
     asserta((1.0-'+')-heavy('A')),
     asserta((1.0-'+')-heavy('B')),
     asserta((1.0-'-')-on_table('A')),
     asserta((0.8-'+')-(on_table(X) <- heavy(X))).
%------------------------------------------------------------

% Atom building predicates

add_prefix(Prefix,Pred,NewPred):-
     name(Pred,PredName),
     append(Prefix,PredName,NewPredName),
     name(NewPred,NewPredName).

% Make (a,...,n) into [a,...,n]
make_list((F,Rest),[F|Rest_ele]) :- make_list(Rest,Rest_ele).
make_list((X),[X]).

variable_free(X) :-
   atomic(X),
   !.
variable_free(X) :-
```

```
   var(X),
   !,
   fail.
variable_free([H|T]) :-
   !,
   variable_free(H),
   variable_free(T).
variable_free(X) :-
   X =.. Y,
   variable_free(Y).



% Output utility predicates.
% ***************************
write_n(_,N):- N @=< 0.
write_n(Char,Num_times) :-
     Num is Num_times - 1, write(Char), write_n(Char, Num).

print_list([]).
print_list([H|Tail]) :- nl,portray_clause(H),print_list(Tail).
% Programming aids
cproj:- [proj],initialize_flags.
ls :- listing(-).
l :- format('~n   **** Facts ****~40| *  C.F. *',[]),
     format('~n~8|=====~43|=====',[]),
     l_facts,
     format('~2n   **** Rules ****~40| *  C.F. *',[]),
     format('~n~8|=====~43|=====',[]),
     l_rules,nl.

l_facts :-
     call((Cf-Sign)-Clause), \+(Clause = (_<-_)),
     format('~n~w ~40|: ~w ~2f .',[Clause,Sign,Cf]),
     fail.
l_facts.
l_rules :- call((Cf-Sign)-(X<-Y)),Clause = (X<-Y),
     nl,portray_clause(Clause),
     format(' ~40|: ~w ~2f .',[Sign,Cf]),
     fail.
l_rules.

% REV/2
% *****
% Arg 1: Original list
% Arg 2: Reversed list
% This predicate sets up the arguments to allow for
%  a difference list reversal of the list
rev(List,Rev_list) :-
     append(List,D,D_list),
     dl_rev(D_list-D,Rev_list-[]).

% DL_REV/2
% ********
% Reverse a list using difference list
dl_rev(A-Z,L-L) :- A==Z.
dl_rev([X|L]-Z,Ra-Rz) :- dl_rev(L-Z,Ra-[X|Rz]).

pop(X,[X-L|T],T,L).
pop(X,[H-L1|T],[H-L1|Rest],L) :-
  pop(X,T,Rest,L).

cf_sort(List,Sorted_list) :-
  keysort(List,Key_list),
  rev(Key_list,Sorted_list).
```

```
print_choice_ls([]).
print_choice_ls([H-L|T]) :-
  portray_clause(H),print_list(L), print_choice_ls(T).

report :-
  retract(answers([(Cf-Sign)-Ans|T])),
  asserta(answers(T)),
  test_cf_ans((Cf-Sign)-Ans).

test_cf_ans((Cf-'-')-A) :- (flag goal_variable_free,on),format('No
~2f ~n',[Cf]).
test_cf_ans((Cf-'-')-A) :- (flag goal_variable_free,off),backtrack.
test_cf_ans((Cf-'+')-A) :- (flag goal_variable_free,on),
     all_answers(All_answers),
     \+(contradictory((Cf-'+')-A,All_answers)),
     \+(more_certain((Cf-'+')-A,All_answers)),
     format('Yes ~2f ~n',[Cf]).
test_cf_ans((Cf-'+')-A) :- (flag goal_variable_free,off),
     all_answers(All_answers),
     \+(contradictory((Cf-'+')-A,All_answers)),
     \+(more_certain((Cf-'+')-A,All_answers)),
     write('Yes '),write(A),write(Cf),wait_user.
test_cf_ans(_) :- backtrack.

contradictory((Cf-'+')-A,List) :- member((_-'-')-A,List).
more_certain((Cf-'+')-A,List) :-member((Cf2-_)-A,List),
     Cf2 @>Cf.

wait_user :- nl,write('Continue (y/n) :'),get(X),do(X).
do(121) :- backtrack.
do(_).

clean :- X-Y, retract(X-Y), clean.
```

```
clean.
killflags :- abolish(flag,1).
add_db(X) :- X.

%*************************************%
% Unification with occurs check from Theorist codE
unif(X,Y) :-
   var(X), var(Y), X=Y,!.
unif(X,Y) :-
   var(X),!,
   \+ appears_in(X,Y),
   X=Y.
unif(X,Y) :-
   var(Y),!,
   \+ appears_in(Y,X),
   X=Y.
unif(X,Y) :-
   atomic(X),!,X=Y.
unif([H1|T1],[H2|T2]) :- !,
   unif(H1,H2),
   unif(T1,T2).
unif(X,Y) :-
   \+ atomic(Y),
   X=..XS,
   Y=..YS,
   unif(XS,YS).

appears_in(_,_) :- (flag occurs_check,off),!,fail.
appears_in(X,Y) :-
   var(Y),!,X==Y.
appears_in(X,[H|T]) :- !,
   (appears_in(X,H); appears_in(X,T)).
appears_in(X,S) :-
```

```
   \+ atomic(S),
   S =.. L,
   appears_in(X,L).
%************************************%

% Programming aids
q :- halt.

% DEBUG Aids
% This debug facility can be turned on and off by setting the
% trace flag on or off.

% show/1
% ******
% This can be place at any point in the program.
% It prints the argument supplied and then prints a
% snapshot of all the lists important to our resolution

show(Pointer) :-
  (flag trace,on),
  write(Pointer),nl,
  goal(G),choice_pts(C),resolvant(R),active_list(A),answers(Ans),
  all_answers(All_ans),
  write('goal : '),write(G),nl,nl,
  write('Choice_pts : '),nl,print_choice_ls(C),nl,
  write('Resolvant : '),write(R),nl,nl,
  write('Active list : '),print_list(A),nl,
  write('Answers : '),write(Ans),nl,
  write('All answers : '),write(All_ans),nl,
  get(X),nl.
show(_) :-
  (flag trace,off).
%#################################################################%
```

Index

VITA AUCTORIS

Viren Parasram was born in 1960 in **New Delhi, India**. He graduated from ***Forster's High School*** on 1983. From there he went on to the **University of Windsor** where he obtained a Bachelor of Computer Science in 1986. He is currently a candidate for a Master's degree in Science at the University of Windsor and hopes to graduate in the Summer of 1994.