Default Reasoning

and

Neural Networks

By

IRENE GOVENDER

DEFAULT REASONING AND NEURAL NETWORKS

by

IRENE GOVENDER

submitted in part fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

in the subject

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: J.H. ROSENBLATT

JUNE 1998

****

CONTENTS

ACKNOWLEDGEMENTS

I would like to thank Helene Rosenblatt for her patience and guidance in leading me through this project. Her encouragement and support has enabled me to see this dissertation through to its completion.

Secondly, I would like to thank my husband, Desmond, for his support and sacrifice, and my two sons Stanton and Joash for their patience during my long hours of occupation with the writing of this dissertation.

The financial assistance of the Centre for Science Development (HSRC South Africa) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the Centre for Science Development.

ABSTRACT

In this dissertation a formalisation of nonmonotonic reasoning, namely Default logic, is discussed. A proof theory for default logic and a variant of Default logic – Prioritised Default logic – is presented. We also pursue an investigation into the relationship between default reasoning and making inferences in a neural network. The inference problem shifts from the logical problem in Default logic to the optimisation problem in neural networks, in which maximum consistency is aimed at. The inference is realised as an adaptation process that identifies and resolves conflicts between existing knowledge about the relevant world and external information. Knowledge and data are transformed into constraint equations and the nodes in the network represent propositions and constraint equations. The violation of constraints is formulated in terms of an energy function. The Hopfield network is shown to be suitable for modelling optimisation problems and default reasoning.

Key terms:

Default logic; normal default theories; Resolution theorem proving; neural networks; penalty logic; Hopfield network; training algorithm; nonmonotonic inferences; expectations; connectionist inference mechanism; conflict resolution strategies

PREFACE

This dissertation gives an exposition of Default logic, which is to a large extent devoted to an investigation of Reiter's [24] theory of default reasoning. Also included is an overview of neural networks and an investigation of modelling default reasoning in neural networks.

Chapter 1 presents a brief discussion of nonmonotonic reasoning in general and default logic is introduced as one of the formalisations of nonmonotonic logic.

Chapter 2 gives a detailed discussion of Default logic supported by a number of examples.

Chapter 3 presents a proof theory for default logic. A variant of default logic, namely Prioritised default logic is discussed based on Mengin's [17] approach. Conflicts among defaults are shown to be resolved using extended clauses and new resolution rules together with classical resolution rules.

In Chapter 4 neural networks are introduced. The functioning of neural nets is compared to the functioning of the human brain and different architectures of neural nets are discussed. Learning algorithms, activation functions, energy functions and weights are defined and discussed. The TSP is discussed as an example of an optimisation problem. In this chapter mathmatical knowledge of equations of motions and partial derivatives is assumed.

Finally in Chapter 5 we describe the relationship between default reasoning and the inferencing in a neural network. This discussion is based primarily on the work of Pinkas [21] who introduced Penalty logic as a framework for representing nonmonotonic logic, and Narazaki and Ralescu [19] who proposed a connectionist inference method for a knowledge base containing default knowledge. A number of examples are given in terms of representation in neural networks.

1 INTRODUCTION

Reasoning is a fundamental component of intelligence. Therefore artificial intelligence requires programs with the ability to reason. To this end the study of models of reasoning is a major focus in artificial intelligence (AI) research.

Earlier studies in artificial intelligence was based on a type of reasoning that could be modelled by classical logic. A set of beliefs are given about the world and with reasoning methods such as mathematical deduction, induction or resolution theorem proving, conclusions are derived which extend the set of beliefs about the world. Previous beliefs can never be retracted. We refer to this type of reasoning as monotonic. Classical logic deals with statements that are wholly true or wholly false. Therefore uncertain knowledge or common sense assumptions cannot be represented in classical logic.

When building an intelligent system, the aim is to make the system resemble the reasoning displayed by humans as closely as possible. However, human reasoning is much more complex than what classical logic can support. Hence another form of reasoning is incorporated in the realm of artificial intelligence, namely nonmonotonic reasoning. This type of reasoning has overcome many of the limitations of classical logic, in that it uses methods of deduction and retraction of beliefs that are closer to human reasoning. It allows assumptions to be made which can later be retracted in the light of newly acquired facts. Cognitive behaviour is the result of much computational activity. In [28] Lokendra Shastri highlights these cognitive behaviour patterns by identifying two different kinds of inference:

1. Reflexive inference, which is almost immediate reaction where the agent is not conscious of the steps involved in arriving at a conclusion. This is a very quick procedure. An example of reflexive inference is when one makes a subconscious decision concerning the meaning of a word.
2. Reflective inference; this type of inference is deliberate and relatively slow. Here the agent is aware of the steps involved in arriving at a conclusion. For example the task of adding 567 to 895.

In order to simulate human reasoning accurately we need to allow for both these types of inference. Clearly human reasoning is a very complex process which is nonmonotonic in nature.

AI strives to emulate the reasoning patterns of humans. Many approaches have been developed, which use a pure logicist view. Theories have been developed to support the nonmonotonic nature of commonsense reasoning and a number of formalisations for nonmonotonic reasoning have been developed. In this dissertation one such formalisation, namely Default logic, will be discussed. We will show how neural networks are able to emulate the basic functioning of the brain. Finally we investigate whether symbolic representations of propositions can be implemented in neural networks and whether these networks can perform the kind of inferences required for default reasoning.

A complete description of Default logic is presented in chapter 2. Chapter 3 describes the proof theory used in default reasoning and the prioritization of defaults (in case of conflict) to acquire a minimal set of extensions. In chapter 4 the charateristics and architecture of neural networks are presented, showing the close resemblance to the human brain. Finally, chapter 5 describes the possibility of simulating default reasoning in a neural network.

2 DEFAULT LOGIC

2.1 Introduction

It often happens that one may reason about a set of beliefs (incompletely specified world) and derive conclusions which are plausible, but which can become contradictory at a later stage when new information come to light. Previous beliefs may need to be retracted, which means that the database of beliefs does not expand monotonically. This type of reasoning is therefore said to be nonmonotonic. Nonmonotonic reasoning makes use of non-sound inferences. We say that an inference procedure is sound if any sentence that can be derived from a belief set using that inference procedure is logically implied by that belief set [24]. In other words, inference rules used in this type of reasoning allows one to make temporary or rather default assumptions which are not necessarily logically implied by the belief set.

In order to apply nonmonotonic reasoning successfully in AI, it needs to be formalised. There are a number of ways of formalising nonmonotonic reasoning. These include circumscription [18], predicate completion [3], the closed world assumption [24] and Default logic [26]. In Default logic, the deduction rules are represented by special linguistic expressions called *defaults*. The rest of this chapter is devoted to Default logic.

2.2 Default rules and Default theories

Before we deal with the formalisation of Default logic in detail, we need to discuss the underlying terminology and concepts.

2.2.1 Defaults

Defaults are rules of the form $\dfrac{A(x):\ B(x)}{C(x)}$, where A,B and C are first-order predicates,

A(x) is the prerequisite, B(x) the justification, and C(x) the consequent.

A default such as,

$$\frac{North(x):Is\_Cold(x)}{Is\_Cold(x)}$$

is interpreted as: for any place x, if x is in the North and it is consistent with our current set of beliefs to assume that x is cold, then add this fact to the belief set. In this default, North(x) is referred to as the *prerequisite*, the top Is_Cold(x) is the *justification* and the bottom Is_Cold(x) the *consequent.*

The justification and the consequent may be different predicates, and the prerequisite can be omitted. In the latter case the only requirement for applicability of the default is that the justification is consistent with the current belief set. The justification may consist of more than one wff (short for well-formed formula), for example, $\dfrac{A(x):B_1(x),B_2(x)}{C(x)}$,

where it would be required that both $B_1(x)$ and $B_2(x)$ are consistent with the current belief set.

2.2.2 Default Theories

The main elements of Default logic are pairs consisting of a set of wffs, the *axioms* (denoted by W), and a set of *defaults* (denoted by D). Such a pair is referred to as a *default theory (T)*, which represents factual and common sense knowledge about the specific world that is involved. We write T = <W,D>, where W is the set of axioms and

D the set of defaults.

Example 2.1

Consider the theory T consisting of the axioms,

Is_Hot(x) and Is_Raining(x) and the default

$$\frac{\text{Is\_Hot}(x):\text{Humid}(x)}{\text{Humid}(x).}$$

Now, W = {Is_Hot(x),Is_Raining(x)} ,

$$D = \left\{ \frac{\text{Is\_Hot}(x):\text{Humid}(x)}{\text{Humid}(x)} \right\} \quad \text{and}$$

T = <W,D>. ■

A default is said to be *open* if it contains at least one free variable, i.e. a variable that does not appear within the scope of a universal quantifier (∀) or an existential quantitfier (∃). For example,

$$\frac{\text{Bird}(x):\text{Can\_fly}(x)}{\text{Can\_fly}(x)}$$

is an open default, since x is free. In other words x is not the x in ∀x or ∃x. A default is *closed* if it contains no free variables. Some examples of closed defaults are:

1. $\frac{:\text{Hot\_in\_Summer}}{\text{Hot\_in\_Summer}}$ This represents the rule that if it is consistent with current beliefs that it is Hot_in_Summer then add Hot_in_Summer to the set of beliefs.

2. $\frac{\text{Bird(Tweety)} : \text{Can\_fly(Tweety)} ,}{\text{Can\_fly(Tweety)}}$ where Tweety is a constant. This represents the rule that if Tweety is a bird and it is consistent to believe that Tweety can fly, then add that belief to the theory.

2.3 Open defaults and their instances

Open defaults represent general inference rules which can be applied to different objects or individuals in the world that we are working with. For convenience sometimes the defaults will be written on one line. As we've said above, a default $A(x):B_1(x),\ldots,B_k(x)/C(x)$ is open if and only if at least one of $A(x), B_1(x),\ldots,B_k(x), C(x)$ contains a free variable. A default theory T is said to be open iff it contains at least one open default.

A *term* is a constant, or a variable, or the result of applying some function of the underlying first-order language to an existing term. A *ground term* is a term that contains no variables. We can obtain an *instance* of an open default by replacing its free variables with ground terms. This means that an instance is a closed default. We can identify an open default with the set of all its instances, obtained by using all the possible ground terms in the language as replacement for the free variables. For now we assume that there are finitely many ground terms. Example 2.2 clarifies this.

By replacing each open default theory T with its corresponding set of instances, we get a closed default theory CLOSED(T) [17], thereby eliminating open defaults. Every open default theory T can thus be mapped into a closed theory CLOSED(T). In this discussion we focus on closed theories in particular.

Example 2.2

Suppose the theory T consists of:

W = { Is_Married(Jim) $\wedge$ Is_Married(Tom) $\wedge$ Single(Joe)}

and

$$D = \left\{ \frac{\text{Is\_Married}(x) : \text{Has\_Children}(x)}{\text{Has\_Children}(x)} \right\}$$

The individuals that exist in the world which is modelled by T, are Jim, Tom and Joe. These individuals, or constants, are the ground terms of this theory. In keeping with the above explanation, one can identify the open default of T with the set of instances obtained by replacing x with Jim, Tom and Joe respectively. Consequently, the corresponding closed theory is:

$$W = \{Is\_Married(Jim) \wedge Is\_Married(Tom) \wedge Single(Joe)\}$$

$$D = \left\{ \frac{Is\_Married(Jim): Has\_Children(Jim)}{Has\_Children(Jim)} \quad \frac{Is\_Married(Tom): Has\_Children(Tom)}{Has\_Children(Tom)} \quad \frac{Is\_Married(Joe): Has\_Children(Joe)}{Has\_Children(Joe)} \right\}$$

The third default will never be applied, since it cannot be inferred that Joe is married and so the prerequisite does not hold. The conclusion that Joe has children is plausible only if he was married. In the world represented in this example single persons are not expected to have children. This is however the domain of people's choices and morals, with which this specific world is not concerned. ■

2.4 Closed default theories

A default theory <W,D> is *closed* iff every default of D is closed [26]. According to [17], a closed default theory, written CLOSED(T) and referred to as the closure of T, can be obtained by the following construction:

(1) If T is closed, then put CLOSED(T) = T, else

(2) Replace T by its Skolemized form (explained below) and denote the resulting theory by T_1.

(3) Let T_2 be the theory obtained from T_1 by replacing each of its open defaults by the set of their instances over TERMS(T_1), where TERMS(T_1) is the set of all ground terms constructable from D.

The Skolemized form of a wff is obtained by replacing all existentially quantified variables with a Skolem function, i.e. a new function of which the arguments are the universally quantified variables preceding ∃ in the wff. When no such variables appear, the Skolem function is a new constant. The newly introduced Skolem function(s) or constant(s) must all differ and must differ from existing functions in T. Finally remove the existential quantifier. For example, consider the wff (∀x (∃y Loves(x,y))). We must remove the existential quantifiers. Here the existential quantifier is within the scope of the universal quantifier. In this case the existential variable y depends on the value of the universal variable x. Therefore we drop the existential quantifier and replace the variable y with a term which is a function of x. We now have ∀x Loves(x,f(x)), where f(x) is a Skolem function.

To get the Skolemized form of a theory T, the axioms in W and the consequents of the defaults in D must all be Skolemized (where Skolemization is applicable).

2.5 The extension of a default theory

Since we cannot know everything about the specific world involved, as implied in the introduction to this chapter, there will be gaps in our knowledge. Our belief set will be an incomplete collection of facts about this world. However, there will arise situations in which inferences have to be made despite the incompleteness of the knowledge base. The introduction of defaults allows us to extend the incomplete knowledge base. This extension of the incomplete theory is the crucial part of Default logic.

A theory T can be extended to form the extension E by including new beliefs (wffs) obtained in one of two ways: they are inferred from axioms in W by applying deduction rules such as modus ponens of classical logic, or they are the result of successfully applying defaults in D. Note that a default is *applicable* with respect to a belief set S iff S logically implies the prerequisite of the default and S does not logically imply the negation of the justification of the default.

Example 2.3

Consider the theory T consisting of the axiom,

Cricketer(Stanton) and the defaults

$$\frac{\text{Cricketer}(x):\text{Tall}(x)}{\text{Tall}(x)} \quad \text{and} \quad \frac{\text{Cricketer}(x):\text{Healthy}(x)}{\text{Healthy}(x)}.$$

By the first default, Stanton is tall; by the second, Stanton is healthy. Since both these defaults can be applied simultaneously, that is, after applying one, the other is still applicable, T gives rise to one extension which is given by

E= {Cricketer(Stanton),Tall(Stanton),Healthy(Stanton)}.

E is a set of sentences closed under logical implication with respect to ordinary logic. Therefore the extension E is more accurately written as

E=Th({Cricketer(Stanton),Tall(Stanton),Healthy(Stanton)}), where Th refers to the logical closure of the sentences in the set. ◘

Let T' be the theory obtained by adding the axiom ¬(Tall(Stanton) ∧ Healthy (Stanton)) to W. This says that Stanton can be either Tall or Healthy, but not both. Now, if the first default is applied, we infer that Stanton is tall, but the application of the second default is blocked. That is, to apply the second default, we must first check that the prerequisite, Cricketer(Stanton), holds. This is fine. The next step is to check whether the justification, Healthy(Stanton), is consistent with the current set of beliefs. So far we have

Th({Cricketer(Stanton),¬(Tall(Stanton) ∧ Healthy(Stanton)), Tall(Stanton)}).

From ¬(Tall(Stanton) ∧ Healthy(Stanton)) and Tall(Stanton), we can infer

$\neg$Healthy(Stanton). Hence the justification of the second default is not consistent with this set of beliefs. If, however, the second default is applied first, we infer that Stanton is healthy, and this blocks the application of the first default in a similar way as explained above. The axiom added, means that either Stanton is tall or he is healthy, but not both. So, applying the two defaults in different orders gives rise to two different extensions of T', namely

E_1 =Th({Cricketer(Stanton), $\neg$(Tall(Stanton) $\wedge$ Healthy(Stanton)),Tall(Stanton)})

E_2 =Th({Cricketer(Stanton),$\neg$(Tall(Stanton)$\wedge$ Healthy(Stanton)), Healthy(Stanton)})

The above discussion points out the following three conditions that must hold for an extension E [3,5]:

(C1) It should contain W, i.e. $W \subset E$.

(C2) It should be deductively closed, i.e. Th(E) = E.

(C3) If $(A:B_1,\dots,B_k/C) \in D$, $A \in E$ and $\neg B_1 \notin E,\dots,\neg B_k \notin E$, then $C \in E$.

These are the closure conditions. In the above example we saw two different extensions emerge that satisfy the closure conditions. This brings us to the formal definition of an extension of a default theory, as given by Lukazewicz [17].

Definition of an extension

Let T=<W,D> be a closed default theory over a first-order language L. For any set of sentences $S \subseteq L$, let $\Gamma(S)$ be the smallest set of sentences from L satisfying the following properties:

(D1) $\Gamma(S) = Th(\Gamma(S))$

(D2) $W \subseteq \Gamma(S)$

(D3) If $(A:B_1,\dots,B_k/C) \in D$, $A \in \Gamma(S)$ and $\neg B_1 \notin S,\dots,\neg B_k \notin S$, then $C \in \Gamma(S)$.

Now, a set of wffs $E \subseteq L$ is an extension of T iff $E=\Gamma(E)$.

Consider example 2.3: The theory (T) gives rise to the minimal set which satisfy the closure properties. If we apply properties D1, D2 and D3, where S=E, we get

$\Gamma(E)$=Th({Cricketer(Stanton),Tall(Stanton), Healthy(Stanton)}) = E which means E is an extension of the theory T.

Consider another example.

Example 2.4

Let W= {Cricketer(Stanton)} and

$$D=\left\{\frac{\text{Cricketer(Stanton):Tall(Stanton)}}{\text{Tall(Stanton)}}\right\}$$

There are two sets which satisfy the closure conditions, C1, C2 and C3 with respect to T.

E_1 = Th({Cricketer(Stanton),Tall(Stanton)})

E_2 = Th({Cricketer(Stanton),¬Tall(Stanton)}).

Applying D1, D2 and D3 with S= E_1 and S=E_2 respectively, we obtain

$\Gamma(E_1)$ = Th({Cricketer(Stanton),Tall(Stanton)}) = E_1

$\Gamma(E_2)$ = Th({Cricketer(Stanton)}) $\neq E_2$

So only E_1 is an extension of T.

To build up $\Gamma(E_2)$, we must first apply D1 and D2 to get Th({Cricketer(Stanton)}). Then we apply D3: There is only one default, namely Cricketer(Stanton):Tall(Stanton)/Tall(Stanton). First we must check whether Cricketer(Stanton) is in $\Gamma(E_2)$. At this stage $\Gamma(E_2)$= Th({Cricketer(Stanton)}), which means the condition $A \in \Gamma(S)$ holds. Now, does the second condition, $\neg B_1 \notin S, \ldots, \neg B_k \notin S$, hold? For our example the condition translates to ¬Tall(Stanton) $\notin E_2$. But ¬Tall(Stanton) does appear in E_2, so we cannot add the consequent of the default to $\Gamma(E_2)$. ◘

While it is important to find an extension which further completes an incomplete database of knowledge represented by a default theory, there are theories which do not have an extension. For example, suppose a theory T consists of the empty set of axioms and one admittedly nonsensical default,

$$d = \frac{:Spherical(Earth)}{\neg Spherical(Earth)}.$$

The following set satisfies the closure conditions C1, C2 and C3: E = Th({¬Spherical(Earth)}). If we apply D1, D2 and D3 to E we get Γ(E)={}. This is because ¬Spherical(Earth) is an element of E (so the last condition of D3 is not met). We see that E ≠ Γ(E) which means that E is not an extension of T. The only other possible extension is E_2=Th({Spherical(Earth), ¬Spherical(Earth)}). Since this set contains a contradiction, the closure consists of all possible wffs. E_2 does not satisfy the closure conditions. In particular condition C3 is not satisfied, since the condition, $\neg B_1 \notin E, \ldots, \neg B_k \notin E$ does not hold. In this example the condition translates to ¬Spherical(Earth) $\notin E_2$. But ¬Spherical(Earth) does appear in E_2. Hence E_2 will not satisfy the definition of an extension, which means T does not have an extension.

Similarly, the next example depicts a theory that has no extension.

Example 2.5

Consider the theory T= <W,D>, where

W = {¬In_hospital(Jack) ∧ Sick(Jack)}

$$D = \left\{ \frac{Sick(Jack) : \neg At\_home(Jack)}{In\_hospital(Jack)} \right\}.$$

In this case the consequent, In_hospital(Jack), of the default denies the axiom. If the default is not applied, then there is no way to derive In_hospital(Jack). But if we follow C3 strictly we are forced to apply the default which, after its application, becomes inapplicable, since the consequent will cause a contradiction in the belief set. To show that T lacks an extension, let's consider the only two possibilities.

E_1 = Th({¬In_hospital(Jack) ∧ Sick(Jack)}) and

E_2 = Th({¬In_hospital(Jack) ∧ Sick(Jack), In_hospital(Jack)}).

Since all three conditions of D3 hold, In_hospital(Jack) is added to $\Gamma(E_2)$, i.e. $\Gamma(E_2)$ = Th({¬In_hospital(Jack) ∧ Sick(Jack), In_hospital(Jack)}). We see that $\Gamma(E_1) \neq E_1$. E_2 contains all possible wffs because it contains a contradiction. So the third condition of D3, i.e. $\neg B \notin E_2$ does not hold and therefore In_hospital(Jack) is not added to $\Gamma(E_2)$. $\Gamma(E_2)$ = Th({¬In_hospital(Jack) ∧ Sick(Jack)}). We observe again that $\Gamma(E_2) \neq E_2$. Therefore neither E_1 nor E_2 are extensions of T, which means that T has no extension.

$\square$

The fact that there exist default theories that do not have extensions, is not very supportive of a general theory of default reasoning. It is therefore desirable to restrict attention to a class of theories which do have extensions [26]. These theories are referred to as *normal default theories.*

2.6 Normal default theories

Reiter [26] defines any default of the form

$$\frac{A(x):B(x)}{B(x)}$$

as being *normal*. That is, where the justification and the consequent are the same.

A *normal default theory* is one in which every default in D is normal. One of the most important properties of normal default theories is that every closed normal default theory has an extension.

Lukaszewicz [17] observes that any closed normal default is a mapping from belief sets into belief sets. He gives a formal definition of this idea as follows:

Let S be a belief set. To each closed normal default d_1 = A:B/B we assign a mapping, denoted by d, given by

$$d(S) = \begin{cases} Th(S \cup \{B\}) & \text{if } d_1 \text{ is applicable wrt S, where B is the consequent of } d_1. \\ S & \text{otherwise.} \end{cases}$$

S is interpreted as the set of beliefs before the application of the default d_1, and d(S) is the set of beliefs after the application of d_1 .

Example 2.6

The default $d = \frac{\text{Sick(Tom):In\_Bed(Tom)}}{\text{In\_Bed(Tom)}}$ is applicable wrt to

S_1 = Th({Sick(Tom)}), and is inapplicable wrt to S_2 = Th({Sick(Jane)}).

So $d(S_1)$=Th({Sick(Tom), In_Bed(Tom)}) and $d(S_2) = S_2$ ◘

We say that S is *stable* with respect to D, iff d(S) = S, for all $d_i \in D$. In other words S cannot be extended by applying defaults from D. In order to determine the required characteristics of extensions for closed normal default theories, we need to discuss the underlying terminology and concepts.

Lukaszewicz [17] defines the term *approachable* as follows:

Let W be a set of sentences and let D be a set of closed normal defaults. Assume further that S is a belief set. We say that S is *approachable from W wrt D* iff, for each belief set S_1 such that $W \subseteq S_1 \subset S$, there is a default $d \in D$ such that $S_1 \subset d(S_1) \subseteq S$. Lukaszewicz [17] then defines the extension of closed normal default theories in the following theorem:

Theorem 2.1

Let T =<T,W> be a closed normal default theory and suppose that E is a belief set. Then E is an extension of T iff

(1) $W \subseteq E$;

(2) E is stable wrt D;

(3) E is approachable from W wrt D.

The proof can be found in [17].

The other important result of closed normal default theories is that they are semimonotonic, which means that adding more closed normal defaults to such a theory will never force the retraction of previously inferred beliefs. That is, if D_1 and D_2 are sets of normal defaults and if E_1 is an extension for the theory $T=<D_1,W>$, then the theory $<D_1 \cup D_2,W>$ has an extension E_2 such that $E_1 \subseteq E_2$. To see why closed normal theories are semimonotonic and not monotonic, we look at a similar situation of adding more axioms to W. If W_1 and W_2 are sets of axioms, and if E_1 is an extension for the theory $T=<D,W_1>$, it does not necessarily follow that the theory $<D,W_1 \cup W_2>$ has an extension E_2 such that $E_1 \subseteq E_2$.

Most default rules can be naturally represented as normal default rules [25]. However, if there are more than one default, their interaction with each other may lead to unexpected or conflicting conclusions. For example, if we have two normal defaults such as A(x): B(x)/B(x) and B(x): C(x)/C(x), then one would like to infer by transitivity that if A(x) then C(x). To show that these interacting defaults can lead to conflicting or rather anomalous conclusions, consider this example.

Example 2.7

Suppose the following defaults exist in a theory.

$$d_1 = \frac{\text{High\_school\_dropout}(x) : \text{Adult}(x)}{\text{Adult}(x)} \quad \text{and}$$

$$d_2 = \frac{\text{Adult}(x) : \text{Employed}(x)}{\text{Employed}(x)}$$

From this one would conclude that "Typically high school dropouts are employed". This is not what we would like to believe. Transitivity has to be blocked. One solution to the problem is to modify the second default so as to block the transitivity.

To this effect the second default is replaced by

$$d_3 = \frac{\text{Adult}(x): \text{Employed}(x) \wedge \neg \text{High\_School\_dropout}(x)}{\text{Employed}(x)}$$

Representing defaults in the above manner avoids inferences that goes against intuition.

Default d_3 is not applicable to persons known to be dropouts. ■

The next section deals with the representation of normal default rules in a way that still maintain the intended interpretation but avoids the anomolous situation. The representation will also lead us to the notion of a semi-normal default theory.

2.7 Semi-normal defaults

Many nonmonotonic rules, such as most of the examples we have seen so far, naturally have the form of a normal default. That is,

$$d = \frac{A(x):B(x)}{B(x)}.$$

It is, however, not the case that all defaults can be rewritten in this form. Some defaults are not in a form that allows this, for example the default in example 2.5. In some cases where it seems as if the defaults can be represented in the normal form, there is still the problem where the interaction between the defaults makes the situation more complex. As we've seen in example 2.7, if the rule which is to be applied interacts with other default rules and/or axioms, then there may be some exceptional circumstances which make the rule application unacceptable. The following example should help to explain this further.

Example 2.8

Consider the following statements:

Tom is poor.

All poor candidates receive bursaries.

Typically, if a candidate receives a bursary, he is intelligent.

This can be represented by the following theory T:

$W = \{Poor(Tom), \forall x\ Poor(x) \rightarrow Receive\_bursary(x)\}$

$D = \{ Receive\_bursary(x): Intelligent(x)/Intelligent(x) \}$

The fact that Tom is poor should not be enough evidence that he is intelligent. But T forces this inference. Hence the transitivity from Poor via Receive_bursary to Intelligent must be blocked somehow, while still preserving the intended interpretation. Lukaszewicz [16] suggests replacing the default by the non-normal default

$$\frac{\text{Receive\_bursary}(x):\text{Intelligent}(x) \wedge \neg\text{Poor}(x)}{\text{Intelligent}(x)}$$ ■

The criterion of applying the replaced default is stronger than that of the original default, thus avoiding an anomalous situation. Depending on the context, the rule may be inapplicable because of the strong justification that is required. In general, the exceptional circumstances are denoted by E(x), and the rule can now be represented as $\frac{A(x):B(x)\wedge\neg E(x)}{B(x)}$.

A default of the form

$\frac{A(x):B(x) \wedge C(x)}{B(x)}$ is

called semi-normal. It is possible to express a single-justification default by means of a semi-normal default, provided that the single-justification default correctly models a commonsense world. According to Lukaszewicz [17] a default theory is semi-normal iff all of its defaults are semi-normal. The class of semi-normal defaults includes normal defaults, since any normal default A:B/B may be identified with the semi-normal default A:B ∧ True/B.

Lukaszewicz [17] investigates the adequacy of semi-normal defaults. That is,

1. Are semi-normal defaults sufficient to cover all practical situations?
2. To what degree can the effects of semi-normal defaults be obtained within the structure of normal representation?

These issues are addressed in [3,4]. There Lukaszewicz proposes a translation of an arbitary single justification default into a normal default in two steps. This schema seems to work for a wide class of defaults.

Step 1 A default of the form

$$d = \frac{A(x):B(x)}{C(x)} \quad \text{is}$$

replaced by the semi-normal default

$$d_1 = \frac{A(x):B(x) \wedge C(x)}{C(x)}$$

Note that the applicability criterion, that is, the justification, is stronger in d_1.

Step 2 The semi-normal default,

$$d_1 = \frac{A(x):B(x) \wedge C(x)}{C(x)}$$

is replaced by the normal default

$$d_2 = \frac{A(x):B(x) \wedge C(x)}{B(x) \wedge C(x)}$$

It is however possible to find practical situations where the second step does not work. The following commonsense example illustrates this point.

Example 2.9

Let $d = \frac{\text{Depressed}(x)\text{: Eating\_disorders}(x)}{\text{Over\_weight}(x)}$

As a first step we replace d by

$$d_1 = \frac{\text{Depressed}(x)\text{: Eating\_disorders}(x) \wedge \text{Over\_weight}(x)}{\text{Over\_weight}(x)}$$

Replacing d_1 by

$$d_2 = \frac{\text{Depressed}(x)\text{: Eating\_disorders}(x) \wedge \text{Over\_weight}(x)}{\text{Eating disorders}(x) \wedge \text{Over\_weight}(x)}$$

can clearly lead to unsupported beliefs. The statement, "Typically a depressed person with eating disorders is overweight", is not acceptable. ■

The following example given by Lukaszewicz [17], shows that there are cases where it can work:

$$d_1 = \frac{Adult(x):Employed(x) \wedge \neg Dropout(x)}{Employed(x)}$$

Applying the translation process, we replace d_1 by

$$d_2 = \frac{Adult(x):Employed(x) \wedge \neg Dropout(x)}{Employed(x) \wedge \neg Dropout(x)}$$

Here the consequent of d_2 is stronger than that of d_1, but still leads to a plausible belief.

The examples discussed above indicate that in general single-justification defaults that model a commonsense setting may be reasonably expressed by means of semi-normal defaults. Transforming semi-normal defaults to its normal representation is plausible only if the the condition "typically, if A(x) and C(x), then B(x) is fulfilled.
We now turn to the crucial aspects of semi-normal default theories.

2.8 Extensions of semi-normal default theories

In the foregoing discussion, among the properties of normal default theories observed, one important result was highlighted, namely that every closed normal theory has an extension. Another important aspect is that closed normal theories are semi-monotonic. Semi-normal default theories, on the other hand, lack these two properties. The lack of semi-monotonicity can be illustrated by the following example taken from [17]:

Example 2.10

Let $T = <\{R(a)\}, \{R(a):P(a) \wedge Q(a)/Q(a)\}>$

T has the extension $E = Th(\{R(a), Q(a)\})$.

By adding the default $:\neg P(a)/ \neg P(a)$, we obtain a new theory which has the extension F=

Th({R(a),¬P(a)}). Clearly $E \not\subset F$, which shows that T is not semi-monotonic. ◘

In other words any belief derivable from a semi-normal default theory T can be invalidated by augmenting T by a new set of closed defaults.

The next example from [17] illustrates the possible lack of an extension of a semi-normal theory.

Example 2.11

Consider the theory T consisting of the empty set of axioms and the following defaults:

$$d_1 = \frac{:p \wedge \neg q}{\neg q} \qquad d_2 = \frac{:q \wedge \neg r}{\neg r} \qquad d_3 = \frac{:r \wedge \neg p}{\neg p}$$

If d_1 is applied then T forces d_3 to be applied, the result of which contradicts the justification of d_1. Similarly if d_2 is applied then T forces the application of d_1, the result of which contradicts the justification of d_2. One would observe that a similar contradiction occurs if d_3 is applied first. ◘

This illustrates that semi-normal theories do not necessarily have extensions. In general the lack of an extension is always the result of a conflict arising from the requirement to apply a default which cannot be applied. In the case of semi-normal default theories only the "non-normal" part of the justification of a default that has been applied would be a potential conflict with other defaults.

Etherington [9] provides a sufficient condition that guarantees the existence of an extension of a subclass of semi-normal default theories. This involves the introduction of new concepts which we shall not pursue any further in this dissertation.

2.9 Semantics for Default logic

Although he did not cover the semantics of Default logic in detail. Reiter [26] gives the following model theoretic definition of an extension: the models of W must be restricted by D, so that those models satisfy all (and only) the wffs of an extension for the theory involved. In this section we pursue this idea by briefly discussing a semantics for Default logic that was first proposed by Lukaszewicz [17]. The discussion below is for the most part based on [27]. We focus on the semantics of closed default theories. In order to understand the discussion some terminology needs to be explained.

2.9.1 Preliminaries

An interpretation I that satisfies a sentence α for all variable assignments is said to be a *model* of α. This means that $I \models \alpha$, iff $I \models \alpha$ for all variable assignments. Let X be a set of first-order models. A formula α is *X-valid* iff α is true in all models in X. A formula α is *X-satisfiable* iff α is true in some model in X. A closed normal default α:w/w is *X-applicable* iff α (the prerequisite) is X-valid and w (the consequent) is X-satisfiable.

2.9.2 A semantics for closed default systems

As mentioned above, Reiter [26] views defaults of a theory <W,D> as restricting the models of W in such a way that:

1) Any restricted class of models of W is the class of all models of some extension of <W, D>.
2) If E is any extension of <W,D>, then there is some such restricted class of models of W which is the class of all models of E.

Since the application of a default yields a possible new axiom (the consequent), defaults can be viewed as operators which transform theories. If an agent chooses a default at each step of reasoning and applies it to the current set of models, X (the belief set), a new set, d(X) (where d is the mapping defined in section 2.6 on normal default theories), results.

The new set of beliefs is the set of all d(X)-valid formulas. If this process is repeated many times, it may happen that the current set of models become stable. That is, for each $d \in D$, $d(X) = X$. This means that X cannot be extended further by applying defaults from D. Each closed normal default can be regarded as a mapping from a set of models into a set of models.

Formally we denote the resulting set of models by $<d_i>X$, where $<d_i>$ is the sequence of defaults being applied. The resulting consequents of the applied defaults are $w_1, \ldots, w_i$. Hence $<d_i>X$ is the set of models for $Th(W \cup \{w_1, \ldots, w_i\})$. The stable set of models for a default theory provides a semantic interpretation for the theory. This brings two important theorems to light:

Theorem 2.2

Let A be a closed normal default theory, and suppose that the set of models Y is stable with respect to A. Then Y is the set of all models for some extension for A.

Example 2.12

Consider the theory A=<W,D>, where

W ={Bird(Tweety)}

D= { Bird(x):Fly(x)/Fly(x)}

The possible world modelled by the theory A, is a model of W. The model of A is X_0 = {Bird={Tweety}, Fly={Tweety}}, there is only one possible variable assignment i.e., x= Tweety. The default is X_0-applicable since the prequisite Bird(Tweety) is true in X_0 and the consequent Fly(Tweety) is X_0-satisfiable.

In this particular case there is just one model in X. $Y = d(X_0) = X_0$ since the consequent is true in all (only one in this case) the models in X_0. The set of models $<d_i>Y$, is stable with respect to A, since $d(Y) = Y$. Furthermore, A has one extension, E= Th({Bird(Tweety),Fly(Tweety)}). Hence Y is the set of model(s) for the extension of A.◘

Theorem 2.3

Let E be an extension for a closed normal default theory A= <W,D> and suppose that X is the set of all models for W, Y is the set of all models for E. Then Y is stable with respect to A.

Let us again consider example 2.12.

E = Th({Bird(Tweety), Fly(Tweety)}). Let Y be the set of model(s) for the extension, E of A. Let X be the set of models of W. As defaults from the set D are applied successfully it becomes true that d(X) = X. Therefore the resulting consequent is w_i of the applied default. It follows therefore that since there is one default in this example, the consequent of d_i that is Fly(Tweety), is represented by w_i . Hence $<d_i>X$ is the set of all model(s) for Th(W $\cup$Fly(Tweety)). It then follows that $<d_i>X= X$. But X = Y, since Y is the set of model(s) for the extension, E. There is just one model in this example. Therefore d(Y)=Y. Hence Y is stable with respect to A.

To summarise: If we have a set of models, X_0, for a theory W, we obtain a semantic characterisation of an extension for W by satisfying the applicable defaults successively. At each step we make the consequent of a default X_i-valid by taking away the models of X_{i-1} in which it is false. Because Default logic for closed normal default theories is semimonotonic, each new consequent added, will be consistent with all those already added.

Lukaszewicz's semantics for default theories is applicable only to closed normal default theories, and he admits that we require a semantics that covers more than normal defaults. He addressed the problem to a certain degree by translating non-normal defaults to normal defaults, as discussed earlier. Still, the Default logic proposed by Lukaszewicz has proved useful. Etherington [9] provides a backtracking mechanism to overcome the problem of Lukaszewicz's approach, which does not always lead to the set of models of an extension. In [9] it is shown that individual defaults are not satisfied independently of the theory in which they occur.

2.10 Conclusion

Default logic is an important formalisation of commonsense reasoning. Determining whether a given default theory has an extension, is the main computational problem which is analogous to testing for validity and making deductions in classical logic. This makes default theories complex, since we are working with prototypical facts rather than "hard" facts about the world. Whenever a new default rule is added to a theory its possible interactions need to be analysed carefully, since it may lead to unacceptable conclusions.

In this chapter we introduced Default logic. We explained the difference between open and closed default theories and described how a set of beliefs can be extended using default rules. A formal definition of extensions was given. We described a restricted class of default theories, normal default theories which has two useful properties that default theories do not have in general. These are that every such theory has an extension and that it is semimonotonic. We concluded the chapter by giving a very brief introduction to the semantics of Default logic. In the next chapter we continue the discussion of Default logic and specifically look at a proof theory for it.

3 A PROOF THEORY FOR DEFAULT LOGIC

3.1 Introduction

A first-order proof in classical logic is a sequence of wffs which are derived as a result of applying rules of inference such as modus ponens and resolution, showing that a given wff (say β) is in the knowledge base. Default logic is a formalisation of nonmonotonic logic that was first introduced by Reiter [26]. In addition to first-order wffs (W), a default theory (T) includes a set of default rules (D) for making default assumptions that may have to be retracted when new wffs that contradict these assumptions are added to the theory. In Default logic the current set of beliefs (which may include retractable beliefs) is represented by some *extension* of the theory. The purpose of a default proof theory would thus be to determine whether a given wff β in the language that underlies the theory, is an element of such an extension.

Finding an *extension* of a default theory is central to Default logic, as an extension gives us a set of all possible beliefs that can be derived from the theory. Once we have an extension, we would typically want to determine whether a given sentence $\beta \in L$ (the language that underlies the theory), is an element of the extension. In determining whether β is in the current belief set (extension), we actually construct a default proof.

Intuitively a default proof of β is a sequence of steps (applications of defaults which are elements of D) that, when applied in the particular sequence, yields the closed wff β. The proof theory described below is specific to closed normal default theories. The formal definition of a default proof of a wff, given by Reiter [10] follows.

3.2 Default proof theory for closed normal default theories

Definition 3.1

Let T= <W,D> be a closed normal default theory and β a closed sentence from L.

A default proof of β with respect to T is a finite sequence $D_0,\ldots,D_k$ of finite subsets of D for which

1. $W \cup CONS(D_0) \vdash \beta$

2. For $1 \leq i \leq k$, $W \cup CONS(D_i) \vdash PRE(D_{i-1})$ for all $1 \leq i \leq k$.

3. $D_k = \varnothing$

4. $W \cup \bigcup_{i=0}^{k} CONS(D_i)$ is satisfiable and consistent with respect to W,

where PRE(D) and CONS(D) respectively denote the prerequisites and consequents of a set D of defaults.

Example 3.1

This example was adapted from Besnard [1].

Consider the closed normal default theory where:

I represents "It is hot".

S represents "The sun is shining".

G represents "Go swimming".

U represents "Use sunscreen", and

$W = \{U \rightarrow I, U \vee S\}$

$$D = \left\{ d_1 = \frac{I \vee S : I \wedge S}{I \wedge S}, \quad d_2 = \frac{I \Rightarrow S : I \rightarrow G}{I \Rightarrow G}, \quad d_3 = \frac{S \wedge G : U}{U}, \quad d_4 = \frac{: \neg S}{\neg S} \right\}$$

Let us see how the above example fits the definition of a default proof of the wff, $I \wedge U$.

Let $D_3 = \{\}$, $D_2 = \{d_1\}$, $D_1 = \{d_1, d_2\}$ and $D_0 = \{d_3\}$.
Since $D_3 = \{\}$ we have $W \cup CONS(\{\}) \vdash PRE(\{D_2\})$, i.e. $\{U \rightarrow I, U \vee S\} \vdash I \vee S$.

Applying the set of defaults $D_2 = \{d_1\}$ yields $W \cup CONS(\{d_1\}) \vdash PRE(\{D_1\})$, i.e. $\{I \wedge S, U \rightarrow I, U \vee S\} \vdash (I \vee S) \wedge (I \Rightarrow S)$.

Next we apply the set of defaults $D_1 = \{d_1, d_2\}$ which yields $W \cup CONS(\{d_1, d_2\}) \vdash PRE(\{D_0\})$, i.e. $\{U \rightarrow I, U \vee S, I \wedge S, I \rightarrow G\} \vdash S \wedge G$. In other words one can see that applying inference rules of classical logic yields the proposition $S \wedge G$.

We then apply the set $D_0 = \{d_3\}$ which yields $W \cup CONS(\{d_3\}) \vdash \beta$ (the goal clause), i.e. $\{U \Rightarrow I, U \vee S, I \wedge S, I \rightarrow G, U\} \vdash I \wedge U$. U is the consequent of d_3. Using modus ponens, we derive I. Therefore $I \wedge U$ is logically entailed by $W \cup CONS(\{d_3\})$.

Finally, we have $W \cup CONS(\{d_1, d_2, d_3\})$ is consistent.

We note that $\{d_3\}$, $\{d_1, d_2\}$, $\{d_1\}$, $\{\}$ is a default proof of $I \wedge U$, since it satisfies all four conditions of the definition. $I \wedge U$ is in the extension $E = Th(W \cup \{U, I \wedge S, I \rightarrow G\})$.
There is another default proof of $I \wedge U$ with respect to <W,D>, which is $\{d_4\}, \{\}$. Again if the definition is applied we find that it is indeed true that $\{U \vee S, U \rightarrow I, \neg S\} \vdash I \wedge U$. ■

The default proof, P, of a wff β is the set(s) $D_0, \ldots, D_k$ of all defaults which are invoked in the proof in a certain order. This set of defaults is referred to the *default support* of P [26]. If we refer to the proof of a specific wff β, we will denote the proof by P_β. We will now give a formal definition of this concept.

Definition 3.2

Suppose that β has a default proof $P_\beta = D_0, \ldots, D_k$. The *default support* of this proof is defined to be $DS(P_\beta) = \bigcup_{i=0}^{k} D_i$.

Some observations with regard to the definition of a default support, DS(P):

The set of defaults invoked in a default proof forms the basis of at least one extension; the fourth criterion of the definition of a default proof of a wff, β, is the same as the requirement that $W \cup CONS(DS(P_\beta))$ is consistent whenever W is. There is no clear method for choosing the sets D_i in deriving a default proof. For instance, how do we choose subsets D_i such that $W \cup CONS(D_0) \vdash \beta$? Do we pick subsets D_i at random and test them? No, a method is required for such a proof, and we will discuss such a method in section 3.4.

Another problem with applying the definition is: how do we verify the satisfiability condition of the last criterion?

3.3 Undecidability of extension membership

In first order proof theory, the validity of a wff is semidecidable, i.e. there is a procedure which will confirm the validity of a wff if it is valid, but the procedure will fail to stop if the wff is not valid. A valid wff is one that is true in every interpretation of the specific world involved. However, the extension membership problem (i.e. given a closed default theory T and a closed wff $\beta \in L$, is there an extension for T which contains β?) is undecidable. In other words, it is not always possible to derive a default proof, even if we know one exists. This is so because the fourth condition of the definition involves showing that some set of wffs is satisfiable (i.e. true in some interpretation). There is no algorithm that can guarantee that it will always verify that a satisfiable wff is indeed satisfiable.

The fact that the proof procedure of a wff is semidecidable for first-order theory and the provability of a wff in Default logic is undecidable shows a major difference between first-order theory and Default logic. In order to compute a proof procedure for closed normal default theories, we have to depend upon a process which is not semidecidable. Hence there will be some heuristics involved in the computation. This heuristic component may sometimes lead to incorrect or mistaken beliefs [26]. This idea impinges upon the computation based on neural networks which is the subject of the next chapter.

With this in mind, Reiter [26] introduces default proofs, both top down and bottom up procedures, using a resolution theorem prover.

3.4 Deriving default proofs

The approach followed here is the one that Reiter uses, namely a top-down approach to find default proofs. It starts with the goal clause to be proved and reasons backwards until a suitable set of premises is reached, which is very much like back-chaining (subgoaling) in conventional proof procedures.

3.4.1 Resolution theorem proving

In classical logic, proof of a sentence β involves the application of inference rules such as modus ponens and the resolution principle. Reiter uses the resolution principle in his proof theory for Default logic.

The basic idea behind resolution is: If we know that P is true or Q is true, and we also know that P is false or R is true, then it must be the case that Q is true or that R is true.

In implementing the resolution inference rule, two important processes must be performed, namely conversion of the wffs involved to clausal form, and unification. A wff is said to be in clausal form if it is a disjunction of wffs, each of which is either an atomic formula or a negation of an atomic formula (If P is an n-ary predicate symbol and $t_1,t_2,\ldots,t_n$ are terms of a language L then $P(t_1,t_2,\ldots,t_n)$ is an atomic formula). Unification is the result of instantiating the variables with terms of the language that underlies the theory involved. The inference rule is then applied by finding in the set of clauses, a pair of clauses which contain complementary literals. This pair of clauses (called the *parent clauses*) are resolved by removing the complementary literals. The remaining literals are then combined disjunctively to form a new clause, called the *resolvent*. If a pair of clauses cannot be resolved because it contains no literals that are complementary, then unification (substituting the appropriate terms for the variables) may make the literals complementary. The resolution rule can then be applied in the usual way. It is important that the particular substitution of variables be maintained in further applications of the resolution rule.

There are a number of resolution strategies that can be used to make the resolution principle more efficient. Using these, redundant or useless inferences are avoided. Some examples of these strategies are *input resolution, unit resolution, linear resolution, deletion strategies and set-of support resolution.* These are first-order proof procedures, and use of any of them requires the wffs to be in clausal form. A procedure to convert a logical expression to clausal form follows.

Firstly, all implication operators, $\rightarrow$, $\leftarrow$, and $\leftrightarrow$ are removed, by substituting equivalent sentences involving only the $\neg$, $\wedge$, and $\vee$ operators. For example, $A\rightarrow B$ is equivalent to $\neg A\vee B$. Next, negations are distributed over other logical operators until each such operator applies to a single atomic sentence. For example $\neg(A\vee B)$ is equivalent to $\neg A\wedge\neg B$. Thirdly, variables are renamed so that each quantifier has an unique variable. This can be illustrated by the following example. $(\forall x\ P(x,x)) \wedge (\exists x\ Q(x))$ can be replaced by $(\forall x\ P(x,x)) \wedge (\exists y\ Q(y))$. In the fourth step we eliminate the existential quantifiers through the process of Skolemization. This has been explained in chapter 2. Thereafter all universal quantifiers are eliminated. The expression is then put into conjunctive normal form, i.e. the disjunction is distributed over the conjunction. This is achieved by repeatedly using the rule: $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$. Finally all operators are removed by writing the conjunction as a set of clauses, e.g. $(\{A,B\},\{A,C\})$. If need be, variables are renamed so that no variable appear more than once within a clause. Let us illustrate the procedure by an example.

Example 3.2

Given the sentence $\forall x\ \exists y[P(x,y) \rightarrow \forall x(Q(x,y) \wedge R(x,y))]$, we convert it to clausal form by applying the above procedure.

1. Eliminate implication: $\forall x\ \exists y\ [\neg P(x,y) \vee \forall x(Q(x,y) \wedge R(x,y))]$
2. Negation already applies to a single atomic sentence.
3. Rename variables so that each quantifier has a unique variable:
   $\forall x\ \exists y\ [\neg P(x,y) \vee \forall z(Q(z,y) \wedge R(z,y))]$
4. Eliminate existential quantifier by introducing Skolem functions since the existential quantifier is within the scope of the universal quantifier:
   $\forall x\ [\neg P(x,F1(x)) \vee \forall z(Q(z,F2(x) \wedge R(z,F2(x))]$
5. Eliminate universal quantifiers: $\neg P(x,F1(x)) \vee (Q(z,F2(x) \wedge R(z,F2(x))$
6. Put the expression into conjunctive normal form:
   $\neg P(x,F1(x)) \vee Q(z,F2(x)) \wedge \neg P(x,F1(x)) \vee R(z,F2(x))$
7. Eliminate operators, that is, write the expression obtained in step 6 in clausal form:
   $\{ \neg P(x,F1(x)), Q(z,F2(x))\}$
   $\{ \neg P(x,F1(x)), R(z,F2(x))\}$
8. Rename variables so that they do not appear in more than one clause:
   $\{ \neg P(x_1,F1(x_1)), Q(z,F2(x_1))\}$
   $\{ \neg P(x_2,F1(x_2)), R(z,F2(x_2))\}$ ◘

Linear Resolution

Reiter uses the linear resolution strategy to find a default proof of a wff. What is the characteristic of a linear resolution proof? A linear resolvent is one in which at least one of the parent clauses is either in the initial belief set or an ancestor of the other parent [12]. A linear deduction is one in which each derived clause is a linear resolvent. Graphically, a linear resolution proof of β from some set of clauses S, is given in the form of figure 3.1 [26]. An explanation follows.

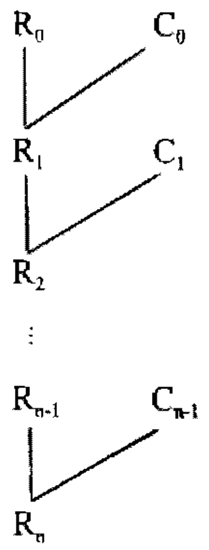


Figure 3.1

One would again ask, how are we going to choose the D_0 so that $W \cup CONS(D_0) \vdash \beta$ (the goal wff that needs to be proved)? The resolution strategy chosen is a top down theorem prover. Therefore the goal wff β can help to choose a suitable subset D_0 of D. Reiter uses figure 3.1 to illustrate the linear resolution procedure. R_0 represents the negation of the goal wff β ($\neg\beta$) and C_0 represents one of the wffs in the initial belief set (W). This satisfies the condition of the linear resolution strategy. R_1 is the resolvent of R_0 and C_0. Next R_2 is obtained from R_1 and C_1, where C_1 is in the initial belief set or an ancestor of R_1. This procedure continues until R_n, an empty set or empty clause is obtained, showing that β is provable by the refutation theorem. This theorem states that If $W \cup \{\beta\}$

is inconsistent, then $W \vdash \neg\beta$, where W is the original belief set [14]. In this case we showed that $W \cup \{\neg\beta\}$ is inconsistent (unsatisfiable) and in the process have demonstrated that W logically implies β ($W \vdash \beta$). Note that R_i are the resolvents and C_i are clauses from the initial belief set or ancestors of R_i, where i = 1...n.

Since this resolution principle is extended to Default logic, we need to represent the default theory T=<W,D> in clausal form. How do we do that? Firstly we write the wffs of W in clausal form according to the procedure above. But we also require that the consequents of the defaults from D be represented in clausal form. Take the default d=(A:B/B) $\in$D, since B is a wff, it can be written in clausal form with B_1 ,...,B_n its clauses. When a default d is applied, we conclude its consequent. But the consequent is in clausal form and there may be more than one consequent clause B_i of the default d. Also there may be more than one default in the theory. To indicate from which default the consequent clause B_i is derived, the consequent clause B_i is indexed by that default. For example, $<B_i, \{d\}>$, $1 \le i \le n$ is called a clausal ordered pair. A pair <C, D>, where C is a clause and D is a set of defaults, is referred to as an *indexed clause*. Now that the details of the representation of the default theory T in clausal form have been specified, we continue with the discussion of a default proof of a sentence β.

We shall use the linear resolution theorem prover to determine the sets $D_0,...,D_k$. The structure of a linear resolution theorem prover for a default theory is represented in figure 3.2 [26].

$$\begin{tikzcd}
(R_0, D_0) \arrow[d, dash] & (C_0, D_0) \arrow[dl, dash] \\
(R_1, D_1) \arrow[d, dash] & (C_1, D_1) \arrow[dl, dash] \\
(R_2, D_2) & (C_2, D_2) \\
\vdots & \iddots \\
(R_{n-1}, D_{n-1}) \arrow[d, dash] & (C_{n-1}, D_{n-1}) \arrow[dl, dash] \\
(R_n, D_n) &
\end{tikzcd}$$

Figure 3.2

We use the principle of refutation to prove the goal clause as we have explained above, using figure 3.1. The difference in figure 3.2 is that now the clauses are indexed, since we need to keep track of the defaults associated with the resolved clauses. Here again the top indexed clause (R_0 , D_0) is such that R_0 is the negation of the clause that we need to prove. (R_i ,D_i) is the resolvent of the indexed clauses (R_{i-1}, D_{i-1}) and (C_{i-1}, D_{i-1}) , for $1 \leq i \leq n$. This process of resolution is repeated until R_n , the empty clause is achieved. In general, if (C_1, D_1) and (C_2, D_2) are indexed clauses, and R is a resolvent of C_1 and C_2, then (R, $D_1 \cup D_2$) is an indexed resolvent of the indexed clauses (C_1, D_1) and (C_2, D_2).

Our next task is to show how the sets D_0, D_1, ..., D_k of Definition 3.1 can be determined using linear resolution. We shall do that by means of an example. This example uses the same theory as Example 3.1.

Example 3.3

Let T=<W,D>, where

W={$U \Rightarrow I$, $U \vee S$} and

$$D = \left\{ d_1 = \frac{I \vee S : I \wedge S}{I \wedge S},\quad d_2 = \frac{I \Rightarrow S : I \Rightarrow G}{I \Rightarrow G},\quad d_3 = \frac{S \wedge G : U}{U},\quad d_4 = \frac{: \neg S}{\neg S} \right\}$$

Let us see how we can use linear resolution to prove the goal clause $I \wedge U$.

Let $\beta = I \wedge U$. Before applying the linear resolution, the theory, $T=<W, D>$ must be represented in clausal form. This yields the following:
$W = \{\neg U \vee I, U \vee S\}$ and the consequent clauses are
$<I,\{d_1\}>$, $<S,\{d_1\}>$, $<\neg I \vee G, \{d_2\}>$, $<U, \{d_3\}>$, and $<\neg S, \{d_4\}>\}$

We start with the negation of the goal clause, $\neg\beta = \neg(I \wedge U) = \neg I \vee \neg U$

$$\begin{tikzcd}
(\neg I \vee \neg U) \arrow[d, dash] & (\neg U \vee I) \arrow[dl, dash] \\
\neg U \arrow[d, dash] & (U,\{d_3\}) \arrow[dl, dash] \\
(\square,\{d_3\}) &
\end{tikzcd}$$

We now have D_0, which is $\{d_3\}$.

Next we need to determine D_1. So we use the prerequisite of the default in D_0 as goal clause, which is $S \wedge G$. Starting with the goal clause and using the linear resolution technique we have:

$$\begin{tikzcd}
(\neg S \vee \neg G) \arrow[d, dash] & (\neg I \vee G,\{d_2\}) \arrow[dl, dash] & \\
(\neg S \vee \neg I,\{d_2\}) \arrow[d, dash] & (I \wedge S, \{d_1\}) \arrow[dl, dash] & \cdots\cdots\cdots\cdots\ (I, \{d_1\}) \text{ and } (S,\{d_1\}) \\
(\square, \{d_1, d_2\}) & &
\end{tikzcd}$$

We now have D_1, which is $\{d_1, d_2\}$.

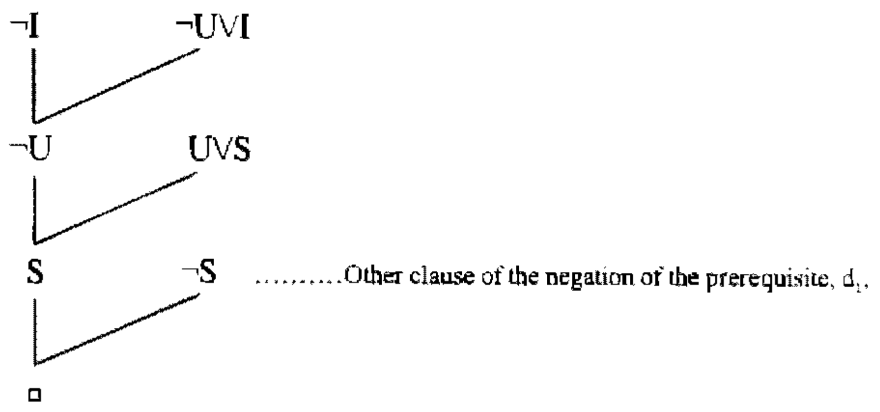
We continue in the same manner, by using the prerequisite(s) of the previous default(s) invoked, as the goal clause. As before we start with the negation of the goal clause. In this case it would be the negation of the conjunction of the prerequisites of d_1 and d_2, i.e. $\neg(I \vee S \wedge I \Leftrightarrow S)$. The prerequisites are not in clausal form. Converting the expression into clausal form yields the following:
$\neg((I \vee S) \wedge ((\neg I \wedge \neg S) \vee (I \wedge S)))$. The negation is taken in to give $\neg(I \vee S) \vee \neg((\neg I \wedge \neg S) \vee (I \wedge S))$. Taking in the negation again gives $\neg(I \vee S) \vee ((I \vee S) \wedge (\neg I \vee \neg S))$. Writing this expression in conjunctive normal form gives us the clauses, $\neg(I \vee S) \vee (I \vee S) \wedge \neg(I \vee S) \vee (\neg I \vee \neg S)$. Now $\neg(I \vee S) \vee (I \vee S)$ gives $(\neg I \wedge \neg S) \vee (I \vee S)$, which in turn yields $(I \vee S \vee \neg I) \wedge (I \vee S \vee \neg S)$. In a similar way $\neg(I \vee S) \vee (\neg I \vee \neg S)$ yields $\neg I \vee \neg S$. The three clauses of the negation of the prerequisites of d_1 and d_2 are $(I \vee S \vee \neg I)$, $(I \vee S \vee \neg S)$ and $(\neg I \vee \neg S)$.

Getting back to the proof, we now have the following:
We start with one of the clauses of the negation of the prerequisites of d_1 and d_2

$$\begin{tikzcd}
\neg I \vee \neg S \arrow[d, dash] & (I,\{d_1\}) \arrow[dl, dash] \\
(\neg S,\{d_1\}) \arrow[d, dash] & (S,\{d_1\} \arrow[dl, dash] \\
(\square,\{d_1\}) &
\end{tikzcd}$$

The procedure returns $\{d_1\}$

The prerequisite of d_1 is $I \vee S$ and its negation is $\neg I \wedge \neg S$. To complete the proof we proceed as before. Again we start with one of the clauses of the prerequisite of d_1.


¬I
¬U∨I
¬U
U∨S
S
¬S
.........Other clause of the negation of the prerequisite, d1.
□

$D_3 = \{\ \}$ is the last set in the sequence that represents the default proof of $I \wedge U$.
The default proof is $\{d_3\}, \{d_1, d_2\}, \{d_1\}, \{\}$. ■

This proof brings us to the following two definitions given in [26]:

Definition 3.3

Let T= <W,D> be a closed normal default theory, where W is a set of ordinary clauses. CLAUSES(T) = $\{<C,\{d\}> \mid d \in D$ and $<C,\{d\}>$ is a consequent clause of $d\} \cup \{<C,\{\}> \mid C \in W\}$

Definition 3.4

A top-down default proof of β wrt T= <W,D> is a sequence $L_0,\ldots,L_k$ of linear resolution proofs such that

1. L_0 is a linear resolution proof of β from CLAUSES(T).
2. For $0 \leq i \leq k$, L_i returns D_i.
3. For $1 \leq i \leq k$, L_i is a linear resolution proof of $PRE(D_{i-1})$ from CLAUSES(T).
4. $D_k = \{\ \}$.
5. $W \cup \bigcup_{i=0}^{k} CONS(D_i)$ is consistent.

The first linear resolution proof of the example above corresponds to L_0, which returns

D_0 of the default proof of $I \wedge U$ (β). L_1 is the linear resolution proof which returns D_1, L_2 is the linear resolution proof which returns D_2 and finally L_3 is the linear resolution proof returning D_3. The elements of CLAUSES(T) include all the indexed clauses, where each clause is indexed by the default which gave rise to it, as well as the ordinary clauses of W.

Some observations can be made with respect to top-down default proofs. At each stage of the proof the current goal, i.e. β or PRE(D_{i-1}) guides the selection of the defaults necessary for establishing the goal. Also noticeable is the fact that a proof of β can be established without invoking any defaults, i.e. using W alone first, and introducing one or more consequent clause(s) only if necessary. If the proof involves only W, and no defaults then β assumes a firmer status. In such a case β is monotonic, in that β need never be retracted under the addition of new first-order facts about the world.

3.5 Soundness and Completeness of default proofs

Linear resolution is sound, which means that a set of beliefs entails β if there exists a linear resolution proof from the set CLAUSES(T) (the set of beliefs together with ¬β), which produces the empty clause. The following theorem states:

Theorem 3.1
Let T =<W,D> a closed normal default theory and β a closed wff. If β has a default proof with respect to T, then β is in some extension E of T.
A complete proof appears in [27] (theorem 3.11).

Linear resolution is also complete for first-order logic. This means that there is a linear resolution proof of a wff β if the set of beliefs entails β. We find that linear resolution is also complete for Default logic. If a set CLAUSES(T) of clauses is unsatisfiable, then there is a resolution deduction of the empty clause {}. This is given specifically in theorem 3.3. The following theorem from [26] (theorem 4.8) describes the completeness property of closed normal default proofs.

Theorem 3.2

Let β be a closed wff. If a consistent closed normal default theory T has an extension E such that $\beta \in E$ then β has a default proof with respect to T.

Since a default proof of a wff β is a set of linear resolution proofs, and a linear resolution proof is an example of a top-down default proof, we can conclude that linear resolution is sound and complete for Default logic. This is given in the following theorem.

Theorem 3.3 [17]

Let T= <W,D> be a consistent closed normal default theory and β a closed wff. Then T has an extension containing β iff there is a top-down default proof of β with respect to T.

3.6 Revision of Beliefs

In chapter 2 it was made clear that generating an extension is actually further completing an incomplete knowledge base. This involves invoking defaults that are coherent with one another for the particular world involved. However it is also clear that there may be more than one extension of a given theory, since there may be more than one combination of coherent defaults. We are at liberty to choose any one of these extensions as our current view of the world. Therefore an agent cannot use just any set of derived beliefs. The beliefs must all belong to a common extension. One would want to use derived beliefs as lemmas for deriving new beliefs. If we do this we will be generating a subset (an approximate extension) of some extension for the original default theory T. This procedure indicates that the derived beliefs $\{\beta_0, \beta_1, \ldots,\}$ must be a subset of some extension E for T. However the following example shows that this condition cannot always be met.

Example 3.4

$$T_0 = \left\langle \{A \rightarrow B\}, \left\{ \frac{:A}{A}, \frac{B:\neg A}{\neg A} \right\} \right\rangle$$

From the theory T_0 we would be able to derive the belief B. This would obviously be β_0, the first derived belief. The new theory now becomes

$$T_1 = \left\langle \{A \rightarrow B, B\}, \left\{ \frac{:A}{A}, \frac{B:\neg A}{\neg A} \right\} \right\rangle$$

Using the new derived B as a lemma, we would be able to derive the belief $\neg A$. This then would be the second derived belief β_1. But $\neg A$ is not in the extension of the original theory T_0 ($E=\{A \rightarrow B, B\}$). ■

Reiter [26] provides a sufficient condition to use derived beliefs as lemmas. This is given by the following theorem.

Theorem 3.4
Let T_0 = <W,D> be a closed normal default theory. In general, suppose T_i has been determined and that $P_{\beta i}$ is a default proof of β_i with respect to T_i . Let T_{i+1} = <$W \cup \{\beta_0,...,\beta_i\}$, D>. For any $n \geq 0$, if $W \cup \bigcup_{i=0}^{n} CONS(DS(P_\beta))$ is consistent then T_0 has an extension E_0 such that $\{\beta_0, ..., \beta_n\} \subseteq E_0$.

However there is still this question: How does an agent proceed if the derived beliefs do not satisfy the consistency property? In other words, can previously held beliefs remain with the addition of new facts about the world? Similar questions can be asked with regard to the addition of defaults. For both the questions posed, Reiter [26] provides conditions in the form of theorems 3.5 and 3.6.

Theorem 3.5
Let T_0 = <W,D> be a closed normal default theory, and let $\{\beta_0, ..., \beta_n\}$ be a set of

derived beliefs determined as stated in theorem 3.4. Suppose further that $F \subseteq L$ is a set of closed wffs. If $W \cup F \cup \bigcup_{i=0}^{n} CONS(DS(P_{\beta i}))$ is consistent then $(W \cup F, D)$ has an extension E such that $\{\beta_0, \ldots, \beta_n\} \subseteq E$.

Theorem 3.6

Let $T_0 = \langle W,D \rangle$ be a closed normal default theory, and let $\{\beta_0, \ldots, \beta_n\}$ be a set of derived beliefs determined as stated in theorem 3.4. Suppose further that D_1 is a set of closed normal defaults. If $\langle W,D \rangle$ has an extension E such that $\{\beta_0, \ldots, \beta_n\} \subseteq E$ then $(D \cup D_1, W)$ has an extension E_1 such that $\{\beta_0, \ldots, \beta_n\} \subseteq E_1$.

However it is still possible that the consistency property fails to hold. This is the subject of a process known as *belief revision*. In [8], Doyle uses a truth maintenance system in order to perform belief revision. We will not dwell on this process in this dissertation. You may refer to Doyle's paper.

It has been stated that there may be more than one set of coherent beliefs, hence more than one extension can be built. An agent cannot work with all the extensions. Therefore one needs to restrict the number of extensions. But how does one do that? One way is to rank the defaults in order of priorities in order to decide which ones should be chosen in case of conflict. This then brings us to the notion of prioritising the defaults described in [19]. The next section describes a general framework for the representation of such priorities, and a mechanism for taking these priorities into account when reasoning by default is investigated.

3.7 Prioritised conflict resolution

A number of authors have proposed an ordering ([2], [15]) among the defaults such that $d_1 \leq d_2$ means that the default d_1 is less reliable than d_2. In effect it means that if d_1 and d_2 are in conflict, then d_2 should be used to generate an extension rather than d_1. In [19], Mengin proposes to associate to such an ordering a so-called *elimination function* – a mechanism for eliminating defaults in case of conflict. He uses this elimination function to represent priorities among the defaults. The purpose of this function is to eliminate one of the minimal elements (those of lowest priority) in order to resolve a conflict. Some preliminary definitions need to be given in order to understand the approach used by Mengin. In [20], he gives another characteristisation of extensions of a default theory, based on the definitions that follow. Mengin uses this characterisation in his approach to conflict resolution.

3.7.1 Preliminaries

Definition 3.5

Let <W,D> be a closed default theory. The *set of formulas generated by a subset U of D*, denoted by $Th^{def}(W \cup U)$, is the smallest set of formulas that contains W, is deductively closed (in the sense of predicate calculus), and such that for all $\frac{f:g}{h} \in U$, if $f \in Th^{def}(W \cup U)$ then $h \in Th^{def}(W \cup U)$.

Example 3.5

Let W = {Cricketer(Stanton), ¬Healthy(Stanton)} and

$$D = \left\{ d_1 = \frac{\text{Cricketer(x):Tall(x)}}{\text{Tall(x)}},\ d_2 = \frac{\text{Cricketer(x):Healthy(x)}}{\text{Healthy (x)}} \right\}$$

$$\text{A subset of D, U} = \left\{ \frac{\text{Cricketer(x):Tall(x)}}{\text{Tall(x)}} \right\}$$

generates the formula Tall(x) after applying d_1. Clearly it holds that for all defaults f:g/h $\in$ U, $W \cup U$ is consistent. Note that d_2 cannot be an element of U otherwise $W \cup U$ would be inconsistent and definition 3.5 would not hold. ■

Definition 3.6 (from [20])

The *conflicts* of a default theory <W,D> are subsets C of D minimal (smallest subset) such that $W \cup C$ is inconsistent. We will denote the set of conflicts of the theory by Ê.

In other words, a conflict is a subset C of D that contains at least one default $\frac{f:g}{h}$ such that $f \wedge \neg g \in Th^{def}(W \cup U)$. The following example illustrates the concepts "conflict" and "minimal".

Example 3.6

Let W = {Cricketer(Stanton), Jockey(Stanton), Cricketer(Joash)} and

$$D = \left\{ \frac{Cricketer(x):Tall(x)}{Tall\ (x)}, \frac{Jockey(x): \neg\, Tall(x)}{\neg Tall(x)} \mid x \in \{Stanton, Joash\} \right\}.$$

The conflict of the theory is the subset

C={Cricketer(Stanton):Tall(Stanton)/Tall(Stanton), Jockey(Stanton): ¬Tall(Stanton)/ ¬Tall(Stanton)}. We say that C is minimally inconsistent since C is the only conflict of the theory. In other words we could not delete any of the defaults from the set C without making $W \cup C$ consistent. We observe that $W \cup D$ is inconsistent, however it is not minimally inconsistent. If we replaced the default variables with the constant Joash then removal of any of the defaults that are unified with Joash would still make $W \cup D$ inconsistent. Therefore $W \cup D$ is not minimally inconsistent. The defaults about Joash are not conflicting, since the prerequisite of the second default cannot be proved (Joash is not a jockey). ◘

Definition 3.7

An *elimination function* on a default theory <W,D> is a function δ that associates, to each conflict C of the theory a set of pairs of the form (d,V), where d is an element of C and V is a consistent subset of $W \cup D$. A subset U of D is *δ-preferred* if it does not contain any conflict of the theory and it verifies:

$\forall d \in D \backslash U, \exists C \in$ Ê, $C \subseteq U \cup \{d\}$ and $(d,U) \in \delta(C)$.

This means that $(d,V) \in \delta(C)$ which implies that in order to resolve the conflict C, d must be eliminated if the wffs in V are true. In the next section a proof theory for Default logic based on the concept of prioritised conflict resolution is given.

3.7.2 A proof theory for Default logic based on prioritised conflict resolution

Keeping in mind that there can be any number of extensions of a theory, we would like to restrict the number of extensions in order to obtain a more 'reliable' world. To do this, Mengin [19] uses priorities among the defaults. Priorities may be based on the ordering of relations among the defaults indicating their reliabilities relative to each other. Checking the coherence of a set of defaults is the same as computing the conflicts of the default theory. In order to compute the conflicts of a default theory, Mengin extends the resolution principle to defaults. He does this by defining:

1. Extended clauses to represent defaults and
2. New resolution rules.

Definition 3.8

The clausal form of a default $d = \frac{\neg CN:CN'}{CN''}$, (where CN, CN′ and CN″ are conjunctions of clauses), is the set of defaults $\bar{d} = \{ \frac{\neg c:c'}{c''}{}_d \mid c \in CN, c' \in CN',$ $c'' \in CN''$, d is the "index" of the default$\}$. $\bar{D}$ will denote the union of the clausal forms of the set of defaults D, and the elements of $\bar{D}$ are called the *extended clauses*.

Some comments can be made with respect to definition 3.8. A default theory T = <W,D> may contain many defaults in the set D. All the prerequisites, justifications and consequents of the different defaults are written in clausal form. The conjunctions of these clausal prerequisites, justifications and consequents respectively, are of the form $\frac{\neg CN:CN'}{CN''}$. Therefore the individual defaults $\frac{\neg c:c'}{c''}$, which are in clausal form, are indexed by d indicating which default the extended clause (default) $\frac{\neg c:c'}{c''}{}_d$ belongs to.

Let us see how these extended clauses are implemented using the new resolution rules proposed in [19]. The new resolution rules has the following format:

$$\frac{\text{Clause/Extended clause} \qquad \text{Extended clause}}{\text{Resolvent(Clause/Extended clause)}}$$

We will use Figure 3.3 to explain the extended resolution rules [19].

Rule 1 $$\frac{c_1 \quad \frac{\neg c_2 : c'}{c''}\, d}{\frac{\neg c : c'}{c''}\, d}$$

Rule 2 $$\frac{\frac{\neg c_1 : c'}{c''}\, d \qquad \frac{\neg c_2 : c'}{c''}\, d}{\frac{\neg c : c'}{c''}\, d}$$

Rule 3 $$\frac{\frac{\neg\square : c'}{c''}\, d}{c''}$$

Rule 4 $$\frac{\frac{\neg\square : c'}{c''}\, d}{\frac{\neg\square : c'}{\top}\, d}$$

Rule 5 $$\frac{c_1 \quad \frac{\neg\square : c_2}{\top}\, d}{\frac{\neg\square : c}{\top}\, d}$$

Rule 6 $$\frac{\frac{\neg\square : c_1}{\top}\, d \qquad \frac{\neg\square : c_2}{\top}\, d}{\frac{\neg\square : c}{\top}\, d}$$

Figure 3.3

The consistency of defaults are checked by applying these new resolution rules, together with the classical rules, to generate an empty clause. Mengin [19] refers to this implementation as the *extended resolution principle*.

In the above figure, c_1 and c_2 denote two clauses whose resolvent is c. When resolutions are made between the prerequisites of the extended clauses, the resolvent is an extended clause (default), or a clause as in the classical sense. Similarly, resolutions are made between the justifications of extended clauses, which results in an extended clause.

Resolutions are also made between the clauses of W and the prerequisite (or justification) clauses of a default. In each of the rules above note that the resolvent is an extended clause or a consequent clause of the extended clause.

In rule 1, resolution is made between the clause c_1 from W and the prerequisite clause $\neg c_2$ of the default. In rule 2 resolution is made between the prerequisites of the two defaults. Rule 3 means that the prerequisite clause is empty and therefore the consequent clause can be used like any wff from W. (In other words the prerequisite clause of d has been proved).

Whenever the extended clause $\frac{\neg\Box : c'}{c''}\, d$ is generated, it means that the prerequisite of this default is proved. To avoid the redundancy of producing the consequent (c'') of d in rule 3 more than once, the result of rule 3 is rewritten in rule 4. This means that the resolvent of rule 3 may be replaced by that of rule 4, provided that the parent clauses are the same for both the resolvents. Observe that the parent default is the same for rule 3 and 4. In rule 5, resolution is made between the clause c_1 from W and the justification clause c_2 of d. Rule 6 indicates the resolution between the justifications of the two defaults.

The clause from W and the consequent clause, which may be obtained after applying rule 3, are used to prove the prerequisite clause of a default d, by applying rule 1 and rule 2. Rule 5 and rule 6 are used to check the consistency of the justification by applying resolution to it. When an extended clause of the form $\frac{\neg\Box : \Box}{\top}\, d$ is produced, it means that the default is in conflict with the theory. A resolution is made between the prerequisites (or the justifications) of two extended clauses if they have been produced from the same default. This can be observed in rules 2 and 6 respectively.

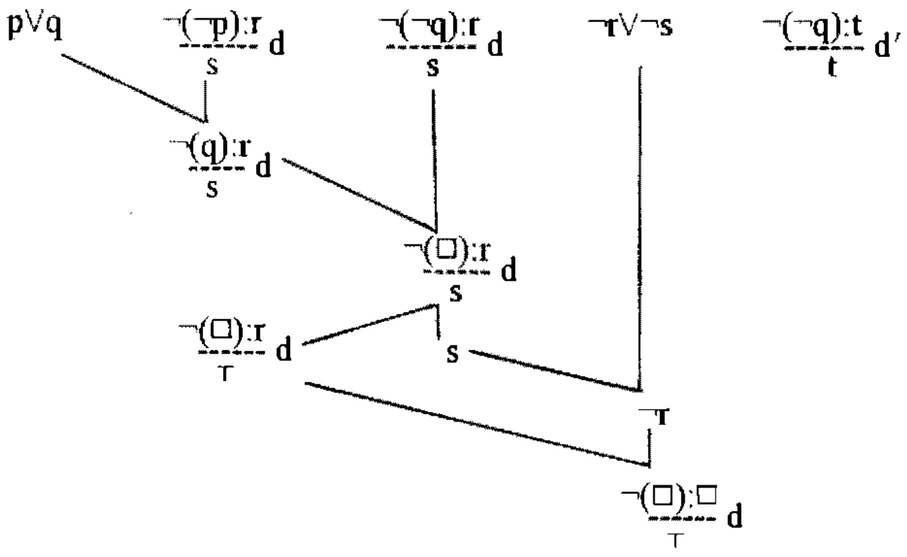
Theorem 3.7 [19]

Let <W,D> be a default theory, such that W is a finite set of clauses and D is finite and can be written in clausal form. A subset C of D is a conflict of the theory iff it is minimal such that the empty clause of the form $\frac{\Box : \Box}{\top}$ d can be produced from $W \cup C$ using the extended rules (1) to (6) and the classical resolution rule.

Consider the abstract example taken from [19].

Example 3.7

Let W={p∨q, ¬r∨¬s} and D = {d, d′}, where d = $\frac{p\vee q:r}{s}$ and d′= $\frac{q:t}{\neg t}$. The resolutions are shown in the figure below, indicating that {d} is a conflict of the theory.



p∨q
$\frac{\neg(\neg p):r}{s}d$
$\frac{\neg(\neg q):r}{s}d$
¬r∨¬s
$\frac{\neg(\neg q):t}{t}d'$
$\frac{\neg(q):r}{s}d$
$\frac{\neg(\square):r}{s}d$
$\frac{\neg(\square):r}{\top}d$
s
¬r
$\frac{\neg(\square):\square}{\top}d$

Figure 3.4

Resolution is made between the wff p∨q of W and the prerequisite of default d using rule 1. (The default d is written as two separate defaults namely ¬(¬p):r/s and ¬(¬q):r/s because p∨q is equivalent to ¬(¬p∧¬q)). The resolvent $\frac{\neg(q):r}{s}d$ and the other prerequisite clause $\frac{\neg(\neg q):r}{s}d$ are then resolved to yield $\frac{\neg(\square):r}{s}d$ using rule 2 (resolution between the prerequisite clauses). Rule 3 is applied to yield the consequent clause s, since the prerequisite clause has been proved according to the extended rules. Also from the same resolvent, we have according to rule 4 the extended default $\frac{\neg(\square):r}{\top}d$. Now s is resolved with the other wff ¬r∨¬s to yield ¬r, by applying the classical resolution rule. Using rule 5 and 6 the justification of the resolvent $\frac{\neg(\square):r}{\top}d$ and ¬r is then resolved to yield $\frac{\neg(\square):\square}{\top}d$ This then proves that {d} is a conflict of the theory. ◘

In [19] Mengin proposes a proof theory for Default logic based on the concept of conflict resolution. The four steps involved in deciding whether a wff β is in at least one extension of the theory <W,D> are:

1 Compute the conflicts of $<W, D\cup\{\beta:a/a\}>$.

2 Compute the elimination function δ on the conflicts of <W,D>.

3 Compute the δ-preferred subsets of D (extensions of the theory).

4 β is in the extension generated by an δ-preferred subset U of D iff there is a conflict $V\cup\{\beta :a/a\}$ of $(W,D \cup \{\beta :a/a\})$ such that $V\subseteq U$.

3.8 Conclusion

For the purposes of this dissertation I will not go into the mathematical detail of prioritising the defaults. This can be read in [19]. However, to summarise Mengin's approach to prioritised Default logic, he provides together with the extension of the resolution principle, a type of deduction theorem that provides a link between the notion of conflicting defaults and that of the monotonic deduction with defaults.

In this chapter a proof theory for Default logic and the notion of prioritized conflict resolution which underlies the basic feature of default reasoning was presented. A theorem prover for Default logic based on the notion of prioritised conflict resolution was investigated, thereby introducing extended clauses and the new resolution rules for this logic. The notion of expectations, which is explained in chapter 5 is closely linked to prioritised conflict resolution. We will see how this notion of expectations may be modelled by neural networks, the subject of chapter 5. But first we need some preliminary background to neural networks in general – the topic of the next chapter.

4 ARTIFICIAL NEURAL NETWORKS

4.1 Introduction

Artificial neural nets are basically mathematical models of information processing. They are modelled by the architecture of the biological neural network – the brain. The secret of the high performance of the human brain is in its massively parallel function and not in the capability of one (unit) neuron. A neural net consists of a large number of processing elements called neurons or units (sometimes called cells or nodes). Neurons can receive input from and send input to other neurons. Each neuron is connected to other neurons by means of directed links with an associated weight. These weights represent information being used by the net to solve a problem. Each neuron has an internal state, called its *activation* or *activity level*. This activity level is a function of the inputs it receives, and its result is sent as a signal through connections to several other neurons.

A neuron can send only one signal at a time, although that signal can be sent to several other neurons. A neuron adjusts its own activity level by summing individual incoming signals scaled down by the weight of the incoming connection. This process is defined by an *activation function*. Activation functions vary from one neural network model to another. Often a sigmoid function is used to make the activity level fall in the range from 0 to 1. This activity level is in fact the output of the neuron. Generally the output will be a discrete value, 0 or 1. The output is 0 until the activity level crosses some *threshold* value when it changes to 1. Positively weighted connections are said to be *excitatory*, and negatively weighted connections are said to be *inhibitory* [6]. In order to understand the workings of an artificial neural net (ANN), one can compare it to the biological neural network. The following diagram is taken from [10].

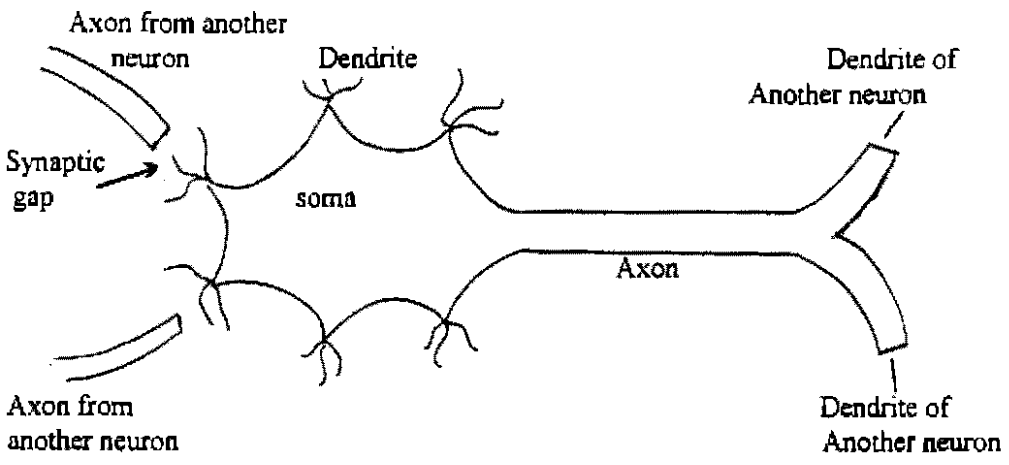


Figure 4.1

Figure 4.1 illustrates a biological neuron, together with axons from two other neurons and dendrites connected to two other neurons. The axons and dendrites are the connections that carry the signals from one neuron to another. The axons allow a neuron to send a signal to other neurons, whereas the dendrites are the connections that allow the neuron to receive the signal from other neurons. The soma or cell body takes cognisance of (sums) the incoming signals from the dendrites. The synaptic gap is the region where one cell excites or inihibits another cell. When sufficient input (synapse strength) is received, (i.e. when some threshold value is reached) the cell is activated, which means that the cell fires. In other words, it transmits a signal over its axon to other cells. Also important is the fact that neurons have failures (die) without negative impact on performance. It is also possible for us to recognize a person in a picture we have seen before or to recognize someone after a long period of time. This characteristic indicates that biological neural networks are fault tolerant. The key features of an ANN, suggested by the properties of the biological neuron can now be summarised:

- The processing element (node) receives many signals.

- Signals may be changed by a weight, similar to that of the action of the chemical process at the receiving synaptic gap (weight is the value associated with a connection path between two processing elements in a neural network).
- The processing element sums the weighted inputs.
- After receiving sufficient input, the neuron transmits a single output.
- The output from a particular neuron may go to many other neurons.
- A weight (synapse strength) may be modified by experience.
- It is fault tolerant, i.e. an ANN can be trained not to take into account small changes to the network, and can be retrained in cases of a lot of damage, e.g. loss of data and some connections [10].

To distinguish ANNs from other approaches to information processing, we need to see how and when ANNs are used.

4.2 How neural networks are used?

There are a number of different types of neural networks. A network is characterized by the following features:

- its pattern of connections between the neurons,
- its method of determining the weights on the connections - this is called its training or learning algorithm, and
- its activation function.

4.2.1 Architecture of an ANN

It is helpful to visualize neurons as arranged in layers. Neurons in the same layer behave in the same manner. Neurons within one layer typically have the same activation function and the same pattern of connections to other neurons in that layer. The neurons within a layer are either fully interconnected or not interconnected at all. If there are more than one layer, then neurons from one layer are connected to neurons in the next layer.

This arrangement of neurons and its pattern of connections are referred to as the net's architecture. Neural nets are generally classified into single layer nets and multilayer nets. When determining the number of layers the input layer is not counted, since it performs no computation in most neural network types. The following figures, taken from [10], illustrate a single layer net and a multilayer net respectively.

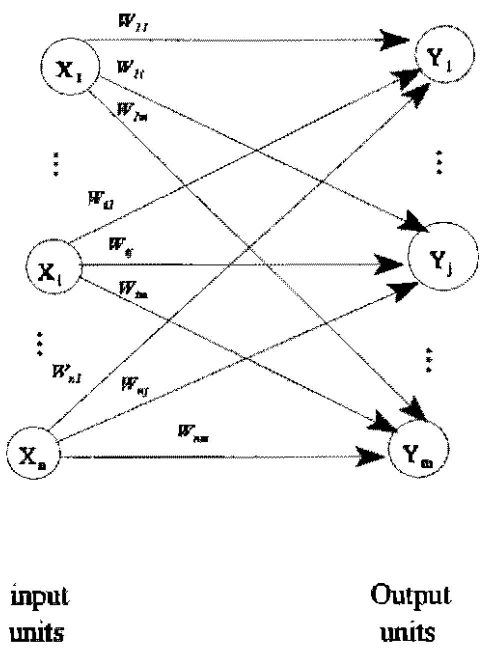


Figure 4.2

An example of when a single layer net is used is when pattern classification is performed, i.e. when classification of vectors (ordered sets of numbers, an n-tuple) in a single category is considered. An input pattern is an example of a vector.

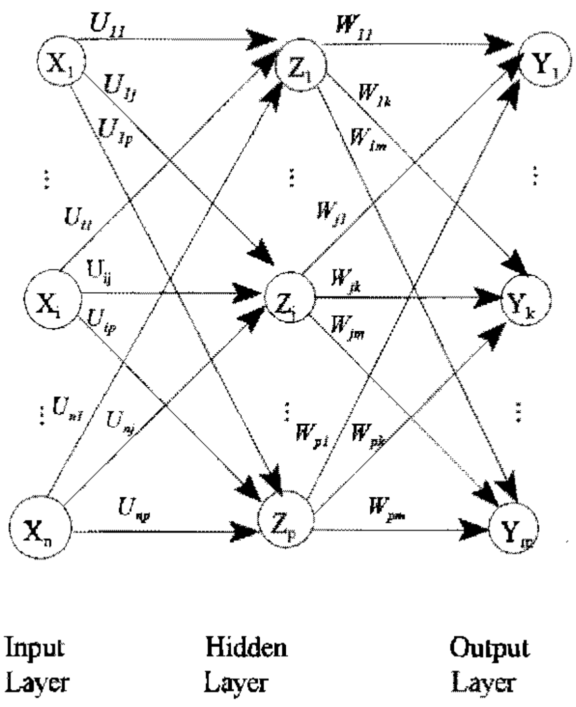


Figure 4.3

Multilayer nets have been used for speech recognition.

The architecture of an associative memory neural net may be feedforward or recurrent (iterative). In feedforward neural nets information flows from the input units to the output units (possibly via intermediate hidden layers of units). Feedforward nets such as the backpropagation net have no connections back to previous layers. In contrast, recurrent (iterative) nets such as the Hopfield net have feedback connections (connections among the units form closed loops).

4.2.2 Training or Learning Algorithm

How do we go about setting the weights on the connections in the net once the architecture is decided upon? The setting and adjusting of the weights is part of the training algorithm required for the net. Learning is synonomous to training in a net and basically involves adjusting weights (synaptic connections) among neurons in such a way that the network acquires the desired behaviour. After training the neural net then produces a representative vector of the output, which is represented on the output nodes.

Three types of training are distinguished, namely supervised, unsupervised and self-supervised training. In deciding on the type of training, one needs to look at the type of problem that must be solved. There is a close correlation between the type of training and the kind of problem.

Supervised training is achieved by presenting a sequence of training patterns or vectors with its associated expected output vectors. The weights are then gradually adjusted according to the learning algorithm, so that the network responds to given inputs with the desired outputs. This type of training is suitable for pattern classification. For example consider the logic AND function:

The input vectors are x_1, x_2

1	1
1	0
0	1
0	0

The target outputs are 1, 0, 0, 0 respectively. In this case there would be two nodes in the input layer and one node in the output layer. After the net has learnt to map the inputs into the desired outputs, the ANN should be able to classify any pair of input vectors correctly.

Unsupervised training is achieved by presenting a sequence of input vectors without associated expected output vectors. The neural net groups similar input vectors together. It modifies the weights so that the most similar vectors are assigned to the same output unit (sometimes referred to as cluster unit) [10]. In other words the network simply plays with input data and tries to discover regularities and relationships between the different parts of the input.

The third type of training method – self-supervised training – can solve constrained optimization problems, that is, problems with conflicting constraints. In other words not all constraints can be satisfied simultaneously. An example of a typical optimization problem is: Given the building of a hospital, what is the best way to lay the wiring for the conduction of electricity for all components requiring electricity? When these nets are designed the weights are set to represent the constraints and the quantity to be maximized or minimized. In trying to quantify the criterion "best" solution in an optimisation problem, a mathematical function (called an energy function) which must be minimised is used. In these cases a nearly optimal solution, which the net can find, is often satisfactory.

4.2.3 Activation function

As stated earlier, the function of an artificial neuron involves summing its weighted input signals. This summation value then goes through a thresholding function, called an activation function. This function decides the node value for the node, i.e. it decides whether the node should be activated or not. In most cases a nonlinear activation function is used. To maintain the advantages achieved when using multilayer nets, as opposed to single layer nets, it is necessary to use nonlinear activation functions. Some of these advantages gained are efficiency, faster computations and the ability to handle more complex problems.

Some common activation functions are:

(*1*) Identity function: $f(x) = x$, this function is for the input units, generally no computation is performed.

(*2*) Binary Step function: $f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$,where θ is a fixed threshold value.

(*3*) Sigmoid function (S-shaped curves), which is continuous and seems more appropriate for nonmonotonic reasoning, since beliefs derived maybe fuzzy. Often a sigmoid function (with range [0,1]) is used as the activation function for neural nets. In these nets the desired output values may be either binary or in the interval [0,1]. For this reason the activation function is called *binary sigmoid.*

The binary sigmoid function is defined as $f(x) = \frac{1}{1+\exp(-\sigma x)}$,
where σ is the steepness parameter.

4.3 The energy function

An energy function is a function of the weights and activations of a neural network. It is monotone nonincreasing and bounded from below. A network is said to reach equilibrium (a stable state) when the energy function settles on a local or global minimum. Each node computes the gradient of the function and modifies its activation value so that the energy decreases gradually. In [13] it has been shown that optimisation problems can be formulated as constraints which are expressed in quadratic energy functions. These functions are minimised using symmetric networks, such as the Hopfield network (described in section 4.4.2 below).

4.4 Models of neural nets

There are many models of neural nets based on their architecture and the learning algorithm. We will briefly discuss two models of neural nets and investigate one with with the view of simulating default reasoning.

4.4.1 Backpropagation neural net

Backpropagation takes input vectors and broadcasts them to the hidden units in the next layer, which in turn takes the activations obtained in each unit and broadcasts it to the next layer until it reaches the output layer. The computed (actual) output is then compared to the target output and this difference (associated error) is propagated back into the net in the reverse direction. Weights are adjusted during this phase. This process is iterated a number of times until some condition is satisfied. The training of the backpropagation net involves three stages: the feedforward of the input training pattern, the backpropagation of the error, and the adjustment of the weights. This type of net implements a learning algorithm for multilayer neural nets based on minimising the mean, or total squared error (the square of the difference between the target and the actual output is summed over all output units and over all training patterns).

4.4.2 The Hopfield network

The Hopfield net is a single large layer of neurons; every neuron connects to every other neuron. The Hopfield network is a memory model with the following features:

- distributed representation (i.e. not one memory location, but a collection of processing elements)
- distributed control (each processing element makes decisions based on its own local situation)
- content addressable memory (a network can store a number of patterns; to retrieve one, we need only specify a part of it)
- fault tolerance (if a few processing elements misbehave or fail, the network will still function correctly).

The Hopfield network processes binary vectors, i.e. vectors with 0's and 1's. However, one can convert the binary vector to an equivalent bipolar vector as follows:

$$v = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \xrightarrow{2v-1} \quad \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \quad (1 \ -1 \ -1 \ 1 \ 1)$$

In 2v-1, 1 denotes a vector with all components equal to 1. The weight matrix (a combination of the vectors of weights) is calculated as follows:

$W = \sum_{k=1}^{L} (2v_k-1)(2v_k-1)^T$, where L is the number of vectors of weights and T denotes the transpose of a vector (rows become columns and vice versa). The net has symmetric weights with no self-connections, i.e. $w_{ij} = w_{ji}$ and $w_{ii} = 0$, for i =

1,.., n and j = 1, ..., n. Therefore the matrix is symmetric. The sum of the inputs into each processing element x_i is

$net_i = \sum_{j=1}^{n} w_{i,j} x_j$, i = 1, 2, ..., n, n is the number of processing elements; w_{ij} is the weight from unit x_i to unit x_j. The sum of the inputs may be written in vector form as net = W.X.

4.4.2.1 The training algorithm in a Hopfield network

To store a set of patterns and their associated output, Fauset [10] implements a rule called the Hebb rule. This rule can be described by the following algorithm;

```
Step 1 Initialize all weights (i = 1,..., n; j = 1,..., n);
        $w_{ij} = 0$.
Step 2 For each input training and target output pair (s:t), do steps 3 to 5.
    Step 3  Set activations for input units to current training input (i = 1, ..., n):
            $x_i := s_i$
    Step 4  Set activations for output units to current target output (j = 1, ..., n):
            $y_j := t_j$
    Step 5  Adjust the weights (i = 1,..., n; j = 1,..., n):
            $w_{ij}$ (new) = $w_{ij}$ (old) + $x_i y_j$ .
```

After the training vectors have been stored in the net, the training begins by implementing the so-called *relaxation algorithm* and the *steady state test.*

Relaxation algorithm

```
Repeat
  1 Choose a random unit $x_i$ (units need not be chosen sequentially, however all
    units must be updated)
  2 Calculate $net_i = \sum_{j=1}^{n} w_{i,j} x_j$ ...... wx is the coefficient of the term in the energy function.
  3 If $net_i > 0$ then $x_i = 1$ ....... Affects the energy function as the x values change.
      Else if $net_i < 0$ then $x_i = 0$
          Else if $net_i = 0$ then $x_i = x_i$
  4 If $x_i$ changes then change := true

Until (all units have been updated);
```

Steady state test

```
Store the input patterns
Enter an input vector
change := false;
Repeat
 For (each input vector) do
   Begin
     1 Run the relaxation algorithm
     2 If change = true then
         steady_state := change (Test for convergence)
   End

Until steady_state is achieved (i.e. no change in the nodes are detected in several
consecutive runs);
```

The energy function, $E(x) = -X^T WX = -X^T.net$

$$= -[x_1, x_2, \ldots, x_n] \begin{bmatrix} net_1 \\ net_2 \\ \cdot \\ \cdot \\ \cdot \\ net_n \end{bmatrix}$$

$$= -\left(\sum_{\substack{i=1 \\ i \neq j}}^{n} x_i net_i + x_j net_j \right)$$

The relaxation process changes an "incorrect" value of x_j to the correct one. For example, if $x_j = 1$ but should be 0, it means that $net_j < 0$ and, therefore $(x_j\ net_j)_{old} < 0$. Relaxation changes x_j to 0, and therefore $(x_j\ net_j)_{new} = 0$. The energy function E(x) is decreased. In a similar way, if $x_j = 0$ but should be 1, it means that $net_j > 0$. The energy function decreases during each step of the relaxation process.

It has been shown in [14] that a large group of logical problems involving real world situations can be structured as optimization problems. Hence, a search for the best solution is required. In order to quantify the vague criterion, "best", a mathematical function (energy function) to be minimised is used.

4.4.2.2 Travelling Salesman Problem (an example)

In [14], Hopfield and Tank draw a parallel between the computational powers used by the nervous systems and the highly interconnected networks of nonlinear analog neurons. The problems to be solved must therefore be formulated in terms of the desired optima, which are often subject to constraints. The classic Travelling Salesman Problem (TSP) is an example of an optimisation problem. The Hopfield network has been used to find the "best" solution. Briefly, a set of n cities C_1, C_2,..., C_n have distances between pairs of cities. The problem is to find a closed tour which visits each city once, returns to the starting city, and has a minimum total path length. As the number of cities increase the time taken to solve the problem grows exponentially. The solution to the TSP, and many optimisation problems is a discrete answer [14]. For the representation of knowledge in a network structure each unit is associated with some meaningful concept and it is proposed in [22] that a "compiler" (program) translate the symbolic representation into a network structure.

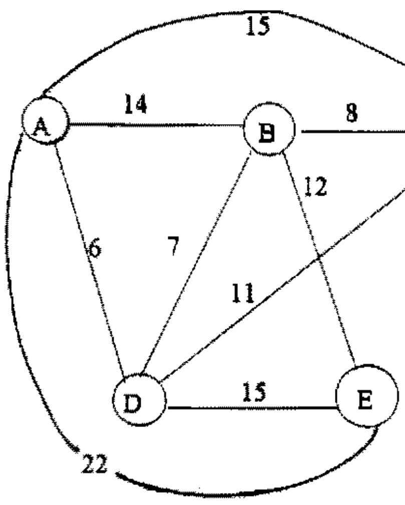
This example will be discussed in detail in order to explain the working of a Hopfield network.

Example 4.1 TSP Problem [14]

In order to solve the TSP problem the following constraints must be satisfied:

- each city is visited only once
- only one city is visted at a time
- the total journey is minimised

We may visualize an example of this problem graphically as follows:



An example of a (non-minimal) path is

A----B----C----D----E----A

14 + 8 + 11 + 15 + 22 = 48 +22

A better solution is

A----D----E----B----C----A

6 + 15 + 12 + 8 +15 = 41 +15

The nodes represent cities and the labels the distances between the cities. In this example there are 5! = 120 routes. How do we describe this problem in terms of a Hopfield network? Let 5 be the number of cities. We will form 5-tuples for each city; for example the 5-tuple for city A is (0, 0, 1, 0, 0) where 1 on the third position means that the city A was visited as the third one. All 5-tuples for five cities will be as follows:

	STOP NUMBER					
City	1	2	3	4	5	
A	0	1	0	0	0	tuple 1
B	0	0	0	0	1	tuple 2
C	1	0	0	0	0	tuple 3
D	0	0	0	1	0	tuple 4
E	0	0	1	0	0	tuple 5

Figure 4.4

This represents the following path between the cities.

C----A----E----D----B----C.

Each cell in the above set of tuples will be represented by one neuron, i.e. the Hopfield layer will have 5 nodes in each tuple, yielding a layer of 25 nodes. Graphically the layer will have a structure as shown below:

tuple 1	tuple 2	tuple 3	tuple 4	tuple 5

The following notation is used:

$V_{x,i}$: output of the neuron corresponding to city x and stop i (xth row , ith column)

$d_{x,y}$: distance between cities x and y.

The network must be described by an energy function in which the lowest energy state (stable state) corresponds to the best path. Basically the energy function must satisfy the following demands:

1. Each city will be visited only once (each row contains only one 1).
2. Only one city is visited at a time (each column contains only one 1).

3. All cities must be visited (the table must contain n 1's. In this example there will be five 1's).
4. The total distance must be the shortest possible.

The first three demands, shown in the form of a permutation matrix which favours stable states, are shown in figure 4.4. Of all the valid tours (5! tours) the energy function must also favour those tours representing short paths.

In earlier work Hopfield [13] showed that the equations of motion for a network with symmetric connections always lead to a convergence to stable states. In [14], Hopfield proposed that dedicated networks of microelectronic neurons could provide the computational capabilities described for a class of problems having combinatorial complexity. He drew a parallel to biological neurons by formulating problems to be solved by an analog computational network. This formulation required the inclusion of seemingly discrete problems in a continuous decision space, in which the neural computation operates. He demonstrates how a continuous decision space and continuous computation can be related to a discrete computation and why a continuous space can improve the solutions obtained by highly-interconnected neural networks. A detailed description of an analog computational network is given in [14]. Here it is shown that the stable states of the analog network consisting of N neurons are the local minima of the energy function. The decision space over which the energy function is minimised is the interior of the N-dimensional hypercube defined by $V_{x,i} = 0$ or 1. The corners (stable states) of this space correspond to the discrete space consisting of the 2^N corners of this hypercube. To summarise, Hopfield showed how "neural" computation of decisions can be obtained in optimisation problems.

The TSP network must be described by an energy function in which the lowest energy state, i.e. the most stable state of the network, corresponds to the best path. The first three demands that the energy function must satisfy can be described by local minima of the energy function.

$$E = A\sum_{x=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{x,i}V_{x,j} + B\sum_{i=1}^{n}\sum_{x=1}^{n}\sum_{y=1}^{n} V_{x,i}V_{x,j} + C(\sum_{x=1}^{n}\sum_{i=1}^{n} V_{x,i} - n)^2$$

where A, B, and C are positive.

The fourth demand that the energy function must also satisfy, that which represents short paths, can be met by adding a term that contains information about the length of a given tour (path). This term is in the form:

$$D\sum_{x=1}^{n}\sum_{\substack{y=1\\ y\neq x}}^{n}\sum_{i=1}^{n} d_{x,y}V_{x,i}(V_{y,i+1} + V_{y,i-1})$$

Hence the total energy function is:

$$E = A\sum_{x=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{x,i}V_{x,j} + B\sum_{i=1}^{n}\sum_{x=1}^{n}\sum_{y=1}^{n} V_{x,i}V_{x,j} + C(\sum_{x=1}^{n}\sum_{i=1}^{n} V_{x,i} - n)^2$$

$$+ D\sum_{x=1}^{n}\sum_{\substack{y=1\\ y\neq x}}^{n}\sum_{i=1}^{n} d_{x,y}V_{x,i}(V_{y,i+1} + V_{y,i-1})$$

We must now construct a weight matrix that corresponds to the above energy function. It is easier to specify what the net should not do than what it should do. Therefore the weight matrix will be defined in terms of inhibitions between neurons. We will index weight by using four indices; $W_{x,i,y,j}$ (connection between neurons x,i and y,j).

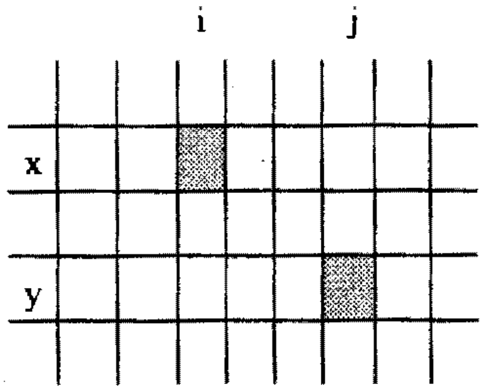


Figure 4.5

We use the energy function and a special function $\partial_{i,j}$ to determine the weight matrix. ∂

is defined as $\partial_{i,j} = 1$ if i=j and 0 otherwise.

We look at how each term of the energy function is used:

Term 1: If one unit in a row fires, it should inhibit other units. It is a kind of a winner-take-all situation. Let us set $W_{x,i,y,j} = -A\ \partial_{x,y}(1-\partial_{i,j})$. x=y on one row, means that $\partial_{x,y} = 0$. If $i \neq j$, this will be equal to -A; the neuron at position i will inhibit others but not itself.

Term 2: If one unit in a column fires, it should inhibit other units in that column in a similar manner as explained above. $W_{x,i,y,j} = -B\ \partial_{i,j}\ (1-\partial_{x,y})$

Term 3: This involves all neurons, so it has a global character. This term is zero if and only if there are n entries of 1 in the entire matrix.We use a global inhibition inhibiting each unit equally: $W_{x,i,y,j} = -\ C$

Term 4: The selection of adjacent cities is inhibited in proportion to their distance. Thus the weight $W_{x,i,y,j} = -D\ d_{x,y}(\underbrace{\partial_{j,i+1} + \partial_{j,i-1}}_{\text{Adjacent columns to Column j}})$

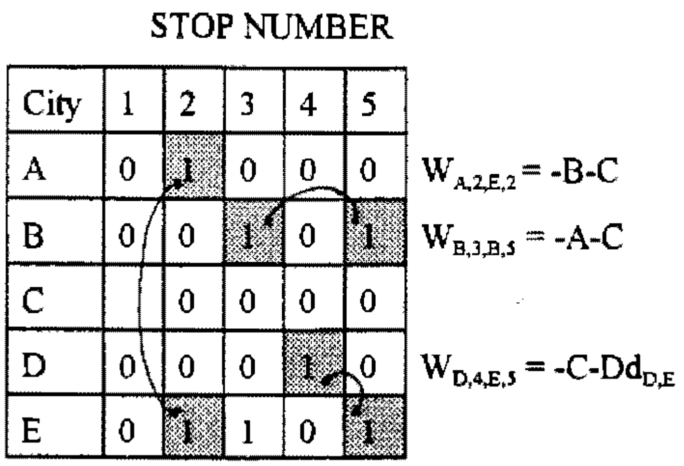
All the above inhibitions will be put together for each weight:

for x = 1,..., n; y=1,...,n; i=1,...,n; j=1,...,n

$$
\begin{aligned}
W_{x,i,y,j} = & -A\ \partial_{x,y}(1-\partial_{i,j}) \\
& -B\ \partial_{i,j}(1-\partial_{x,y}) \\
& -C \\
& -D\ d_{x,y}(\partial_{j,i+1} + \partial_{j,i-1})
\end{aligned}
$$

($\partial_{i,j} = 1$ if i=j and 0 otherwise)

We view the above scenario in the diagram that follows.



STOP NUMBER

City	1	2	3	4	5
A	0	1	0	0	0
B	0	0	1	0	1
C		0	0	0	0
D	0	0	0	1	0
E	0	1	1	0	1

$W_{A,2,E,2} = -B-C$

$W_{B,3,B,5} = -A-C$

$W_{D,4,E,5} = -C-Dd_{D,E}$

Figure 4.6

The terms with A, B, and C coefficients are the general constraints required for any TSP problem. This energy function guides the dynamics of the circuit. The "data term" with coefficient D controls which one of n! sets of constrained final states is actually chosen as the best route. The relaxation algorithm and steady state test is implemented, using the computer. In [14] the simulation results of the TSP problem is given. These results indicate that the coefficients must be set appropriately and in this case it was found that these coefficients were set as follows:
A=B=500, C=200, D=500. ◘

4.5 Default logic and Hopfield Networks

It has been shown in [14] that optimisation problems can be solved satisfactorily using the Hopfield neural network. We have said in chapter 3 that default knowledge (default rules) can interact with each other and may cause conflicts. Therefore the problem of making a decision or finding the best possible route, taking into account the pieces of knowledge and the default rules present, can be seen as an optimisation problem. In other words the constraints of the problem must be satisfied simultaneously. Narazaki and Ralescu [21] has shown that default knowledge can be represented in a neural network. Their work has been motivated by the Hopfield network. Basically knowledge such as that contained in IF-THEN rules (logical implication) are translated into constraints. This will be discussed in chapter 5. Because of the uncertainty of the pieces of default knowledge and the macroscopic (where the whole knowledge base is considered) nature of the knowledge, the continuous Hopfield network is used instead of the discrete Hopfield network. Sometimes conclusions are drawn that are considered fuzzy, therefore a continuous network would be more suitable.

4.6 Conclusion

In this chapter the basic architecture and training algorithm of neural networks have been presented. A parallel between the human brain and the neural network has been shown. A well defined optimisation problem namely the Travelling Salesman Problem (TSP) has been presented. The simulated results of the TSP given in [14] indicate that the Hopfield network is an appropriate network to model optimisation problems. This network exploits parallelism and is considered to be a promising route to effectively emulate the human brain. We will pursue this in chapter 5.

5 NONMONOTONIC INFERENCE IN NEURAL NETWORKS

5.1 Introduction

Classically an agent's knowledge is represented as a collection of well-formed formulas. Most AI reasoning systems are based on this logicist view. That is, logic formulas and a formal proof theory are used to reason about beliefs and infer new knowledge. The advantage of this approach is its ability to fit all predicates, propositions and or axioms into a neat framework of symbolic representation. Classical logic systems are, however, inadequate for modeling commonsense reasoning, therefore default reasoning cannot be adequately catered for in classical logic. This was discussed in chapters 2 and 3.

This chapter investigates default reasoning using neural networks. Because of its architecture of connections, using neural networks is generally referred to as the connectionists approach. This approach concentrates on powerful learning and adaptation mechanisms, as opposed to research that concentrates on the development of powerful knowledge representation systems. A number of researchers have worked in this field, however, the limited scope of this dissertation does not justify covering all the related research. Our discussion is to a large extent borrowed from Narazaki and Ralescu [21], [22] and Pinkas [23].

5.2 Using expectations in default reasoning

In providing a proof for a wff β in nonmonotonic logic, the information used in the inference consist not only of what we firmly believe, i.e. the current belief set, but also information about what we *expect* in a given situation. However, an inference procedure for a wff β in classical logic depends entirely on the current belief set. For example, if we know that it is a hot day, we expect the day to be sunny. Therefore this expectation is not a *full belief* but rather defeasible. This means that if it is hot we may go outside to find that it is overcast. So our default assumption that it is sunny should be retracted. Formally, if the premise α is in conflict with some of the expectations, then these expectations are not used when determining whether a wff β follows from α.

Expectations are treated as full beliefs for the purpose of making inferences. Since expectations have different strengths, these expectations or beliefs will not be applicable all the time. Expectations can be viewed as the result of *learning processes*. We could therefore say that expectations are a way of summarising previous experience in a cognitive way [11]. Gärdenfors [11] summarises the idea of expectations in relation to nonmonotonic inference: "α nonmonotonically entails β iff β follows logically from α together with as many as possible of the set of our expectations as are compatible with α". The following example illustrates the concept of "as many as possible of the set ...".

Example 5.1

Assume the following predicates and the axioms (expectations) associated with them:

Healthy(x)

Retired(x)

Works(x)

Expectations: $Healthy(x) \rightarrow Works(x)$ and

$Retired(x) \rightarrow \neg Works(x)$ and

$\neg(Healthy(x) \wedge Retired(x))$

From the set of axioms we can conclude that if all we know about a person is that he is healthy, then we expect that person to work; and if all we know about another person is that he is retired, we expect him not to work. Now if we learn that Peter is healthy and retired, i.e. (Healthy(Peter) $\wedge$ Retired(Peter)), what can we conclude regarding his work? This information is inconsistent with the set of expectations. If we therefore follow Gärdenfors's definition of expectation in relation to inference, we should look for a subset of these expectations that contains as many elements as possible which are consistent with Healthy(Peter) $\wedge$ Retired(Peter).

To show that these expectations are defeasible, Gärdenfors advocates three possible choices.

1. Retain Healthy(x)$\rightarrow$Works(x) and give up Retired(x) $\rightarrow$ $\neg$Works(x). This would mean that Healthy(x) $\wedge$ Retired(x) $\mid\sim$ Works(x), where $\mid\sim$ denotes "nonmonotonically entails".
2. Retain Retired(x) $\rightarrow$ $\neg$Works(x) and give up Healthy(x) $\rightarrow$ Works(x). This would mean that Healthy(x) $\wedge$ Retired(x) $\mid\sim$ $\neg$Works(x).
3. Give up both Healthy(x)$\rightarrow$Works(x) and Retired(x) $\rightarrow$ $\neg$Works(x). In this case we cannot conclude anything concrete about x. ◘

The example indicates that there is no unique solution. However, a choice has to be made and this choice cannot be made by logical considerations alone. We can decide that the belief of the second expectation that retired persons do not work is stronger than the expectation that healthy people work. (By implication retired people would have already worked formally and once retired they will not work again). Healthy people do not necessarily work. (For example, children or mothers at home). In this case one would opt for the second option.

The discussion above indicates how nonmonotonic inferences may be interpreted in terms of underlying expectations. The approach Mengin follows in [19] to resolve conflict is based on prioritised conflict resolution and this ties in with the notion of expectations. Just as an ordering is defined among the defaults, an ordering is defined among the expectations. The expectations are in fact the defaults of the belief set. Therefore the expectations listed in example 5.1 can be written in Default logic format as follows:

Healthy(x): Works(x)/ Works(x)

Retired(x) : ¬Works(x)/¬Works(x)

Healthy(x) : ¬Retired(x)/¬Retired(x)

We now need to show how the prioritized defaults can be modelled by neural networks. In [28] Shastri shows that the concept of expectation can be interpreted in terms of reasoning in neural networks. Since we have shown the equivalence between expectations and defaults it is therefore natural to conclude that defaults can be modelled in neural networks. This is the topic of the sections that follow.

5.3 Nonmonotonic inferences in a neural network

A major difference between the connectionist approach and symbolic knowledge representation approach is that symbolic systems need an interpreter (theorem prover) to process the information represented and to reason with it, whilst connectionist networks do not have such an interpreter. In the latter, the interpreter and the control mechanism are included in the knowledge that is being represented.

In order to use a neural network to simulate nonmonotonic inference, it must be capable of capturing human intuitions such as that found in default reasoning, and must therefore have the ability to learn, incorporate new knowledge, and dynamically change the knowledge base.

In [23] Pinkas extends propositional calculus so that it can accommodate the characteristics of nonmonotonic reasoning. We show how the notion of expectations can be represented using this extended propositional calculus. Pinkas refers to this calculus as *penality logic*[1]. A positive real number is assigned to every belief. This number, called

[1]Our discussion of penalty logic is to a large extent borrowed from Pinkas [23].

a penalty, may be interpreted as a priority of a belief. This logic can cope with inconsistency in the knowledge base as well. It will be shown that every formula in this extended logic can be compiled into a symmetric network.

5.3.1 Penalty logic

Before an example is presented, we look at the formal definition of penalty logic

Definition 5.1

A Penalty Logic wff (PLOFF) φ is a finite set of pairs. Each pair comprises a real positive number, called the penalty, and a propositional wff, called the belief, i.e.

$\varphi = \{<p_i , \Phi> \mid p_i \in R, \Phi \text{ is a wff}, i=1...n\}$.

The following example, adapted from [23] explains the use of penalties.

Example 5.3

Penalty	wff	
1000	H	Is Healthy
1000	R	Is Retired
10	$R \rightarrow \neg W$	Retired persons tend not to work.
10	$H \rightarrow W$	Healthy people tend to work.
3000	Tom	Tom is the person we reason about.

Graphically the above scenario may be represented as shown below.

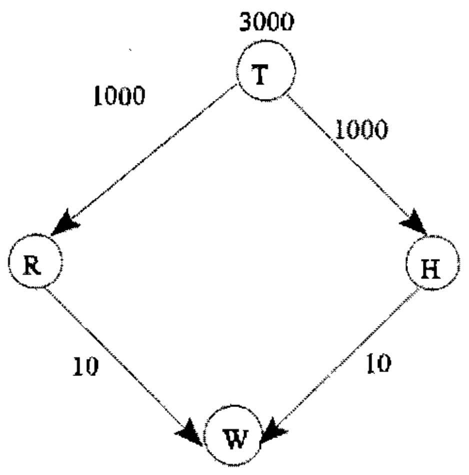


Figure 5.1

The nodes represent the concepts (atomic propositions) work, healthy, retired, and the person we reason about. The links with the arrows between the nodes would represent the proposition. In other words the graph is an inheritance network. The penalties reflect how strong the beliefs are. The higher the penalty, the higher the priority. The concept of priority corresponds to the concept of defeasibility of the beliefs. Higher penalties are assigned to facts, while lower penalties are assigned to defeasible rules such as, "Retired people tend not to work". When we know that somebody is retired, by default we conclude that the person does not work. However, this default assumption can be blocked if we have an exception to the rule. The last fact, T, indicates that Tom is the person we reason about. It is assigned the highest penalty. In this context, Tom would represent *evidence*, and it is considered a temporary (Tom may die or we may reason about someone else) but certain fact. Therefore evidence is not usually part of the knowledge base [23].

We can infer that Tom is both retired and healthy, but what can we deduce regarding his working situation? There is actually no sufficient evidence to believe either W or $\neg$W; in this case W is considered flexible (indecisive). Going back to the ordering of the expectations, in the example, it is more natural to assign a higher penalty to the fact that retired persons do not work than to the fact that healthy people work. The knowledge base is modified as follows:

1000	H
1000	R
15	$R \rightarrow \neg W$
10	$H \rightarrow W$
3000	Tom

There are two competing arguments: one supports the assumption that retired people tend not to work and the other supports the assumption that healthy people work. Now that the penalty has been adjusted, the first argument wins.

5.3.2 A general interpretation of the use of penalties

A knowledge base of assumptions and penalties, $\varphi=\{<p_i, \Phi>\}$, determines a ranking over the set of all possible models for the set of wffs we're working with. (A model of a set of wffs is a truth assignment, i.e. a function from the set of wffs to {T,F}, in which all the wffs are always true.) This ranking reflects the order of typicality we associate with the possible models of this world (some models are more acceptable than others).

Specifying φ intuitively implies that models that satisfy many "more acceptable" assumptions are more appropriate than models that satisfy fewer or "less acceptable" assumptions. Two models may be compared by considering the assumptions in φ that are contradictory. A model is considered "more acceptable" than another model if the sum of the penalties of the contradicting assumptions of the first is less than that of the second. The penalties induce a ranking function that assigns a real value to each of the possible models. This ranking function is called the *violation rank* of φ. The *violation rank* of a PLOFF φ is the function Vrankφ that assigns a real-valued rank to each of the truth assignments in φ. The violation rank for a truth assignment x is computed by summing the penalities for the assumptions of φ that are violated by the assignment. In mathematical terms this means that

$\text{Vrank}\varphi\ (x) = \sum_i p_i\, H_{\neg\Phi i}(x) = \sum_{x \cdot \Phi i} p_i$, where H is the characteristic function of the sub-formula Φ_i. If x does not satisfy Φ_i then x satisfies $\neg\Phi_i$. In this case $H_{\neg\Phi i}(x) = 1$.
If x satisfies Φ_i then $H_{\neg\Phi i}(x) = 0$.

A *preferred model* (a more plausible model) is therefore a model that minimises the Vrankφ function. A PLOFF φ semantically entails Φ, i.e. $\varphi \models \Phi$ iff all the preferred models of φ satisfy Φ.

Example 5.3 (continued)

There are only two models: (TRH¬W) and (TRHW). In the model (TRH¬W) T, R, and H are true and W is false. In the model (TRHW) T, R, H and W are true. Both these models are preferred models, since their violation ranks both have the minimal value 10. There are some valid conclusions as well, namely (T, H$\wedge$R). These conclusions are satisfied by both the preferred models. But we observe that W holds in one preferred model while ¬W holds in the other. ◘

In [23] Pinkas developed a sound and complete proof theory for penalty logic. He ranked consistent subsets of the assumptions of φ, and used the preferred or rather 'best' consistent subset to perform inference. A deduction is made in this proof theory iff all the preferred consistent subsets entail it. A subset T of the assumptions in φ is called a theory of a PLOFF φ iff T is consistent (not contradictory). The penalty of a theory T of φ is the sum of the penalties of the assumptions in φ that are not included in T, i.e. φ-T. The penalty function of φ is defined as $\text{Penalty}_\varphi(T) = \Sigma_{\phi i \in (V\varphi - T)}\, p_i$. The following more formal definition of preferred theories is given in [23].

Definition 5.2

A preferred theory of φ is a theory T that minimizes the Penalty function of φ; that is, Penaltyφ (T)= min{Penaltyφ(S) |S is a theory of φ}.

Example 5.3 (continued)

The set of assumptions (H, R, T, W, ¬W) leads to a contradiction. But there are 2^4 (only 4 propositions, which can take on only one of two values) non-empty consistent subsets of Φ, where at least one belief of φ is missing. Some of the subsets are {T},{T→H}, {T→R}, {H → W}, {T, T→H}, { H → W} (Applying the above discussion, we have the preferred theories, T_1 = {(3000, T), (1000,T→H), (1000,T→ R), (10, H → W)} and T_2 = {(3000,T), (1000, T→ H), (1000,T→R), (10, R→ ¬W)}. To rank the consistent subset T1, we sum the penalties of the missing beliefs of φ. In this case only one belief is missing, namely R→¬W which has a strength of 10, hence the preferred theory T1 is ranked 10. In a similar way it can be shown that the preferred theory T2 also has a rank of 10. Intuitively we understand the reasoning process to be a competition among the consistent subsets. The subsets that win will obviously be the theories with minimal penalty. ◘

Now we turn to the representation of penalty logic in connectionist networks, in particular symmetric connectionist networks.

5.3.3 Representing penalty logic in a neural network

Pinkas [23] describes a PLOFF φ in terms of an energy function as follows:

Definition 5.3

A PLOFF φ is strongly equivalent to an energy function (E) (a function that is bounded below and is a nonincreasing function of the state of the system) iff the violation rank of φ is equal to the characteristic function of E (up to a constant difference).

Before we turn to the representation of a PLOFF in a network, we need to discuss the role the energy function plays in neural networks and in a PLOFF. The updating of the neurons allows a function, called an energy function, to decrease gradually. The crucial aspect of symmetric network models is finding the minima for quadratic functions. This paradigm is used for parallel constraint satisfaction. These models are characterised by a recurrent network architecture, a symmetric matrix of weights (with zero diagonal) and a quadratic energy function that must be minimised. Each node computes the gradient of the function and modifies its activation value so that the energy decreases gradually. We say that the network reaches equilibrium when the function settles on either a local or global minimum. We refer to this state as a stable state (see chapter 4).

There is a direct mapping between the symmetric networks and the quadratic energy functions they minimise. Hence a quadratic energy function can be translated into a corresponding network and vice versa. Weighted arcs in the network correspond to weighted terms (the weights on the arcs are the coefficients of the terms) of two variables in the energy function with opposite sign (see example 5.3 on page 79). Thresholds of units in the network correspond to single variable terms in the function. Figure 5.2 is an example of a network. Its energy function follows.

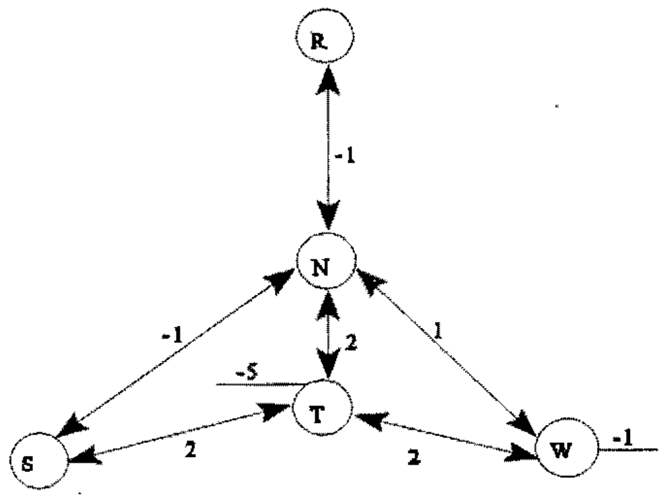


Figure 5.2

This symmetric network represents the function:

E = -2NT-2ST-2WT+5T+NS+RN-WN+W, where T is a hidden unit. In [23] Pinkas describes a way to convert an energy function to contain no hidden variables. He goes on to show that for any such function E with no hidden variables, there exists a corresponding satisfiable wff. For the mathematical detail of this conversion procedure, we refer the reader to [23]. The corresponding energy function containing no hidden variables is E = -NSW + NS +RN - WN +W.

Now rewriting E yields, NS(1-W) + RN + (1-N)W. Using the characteristic function we have E = $H_{N\wedge S\wedge\neg W}$ + $H_{R\wedge N}$ + $H_{\neg N\wedge W}$, which in turn yields

$H_{\neg((N\wedge S)\rightarrow W)}$ +$H_{\neg(R\rightarrow(\neg N))}$ + $H_{\neg(N\vee(\neg W))}$

Hence the wff is (N∧S →W) ∧ (R→(¬N)) ∧ (N∨(¬W)).

In order to represent an arbitrary logic formula, hidden units or the power of high order connections are required in a network. The variables of an energy function can be divided into two sets: visible variables and hidden variables. The hidden variables and visible variables correspond to the hidden units and the visible units respectively, in the network. E(x,t) denotes an energy function with both hidden and visible variables, where x represents the visible variables and t represents the hidden variables. An agent is usually not interested in the value of the hidden units. However, the visible units are used as inputs and outputs, and hence is of interest to the agent. He encodes the input units and then observes the values of the output nodes after the training has occurred or after the network has settled on a stable state. These values are decoded and interpreted to yield an answer. The *characteristic* function of a network is defined to be $Erank_E(x)$ = $\min_y\{E(x,y)\}$, where the characteristic function is the energy function with the minimum number of hidden variables. The Erank_E function characterises the network's behavior, since it defines the energy of the visible states (that is, the energy level obtained when the visible units are assigned the states' values).

Getting back to the representation of penalty logic in a symmetric network, we have in [23] the following theorem:

Theorem 5.1

For every PLOFF $\varphi = \{<p_i,\Phi_i>\} \mid i = 1...n\}$ there exists a strongly equivalent quadratic energy function $E(x,t)$, that is, there exists a constant c such that $Vrank_\varphi = Erank_E + c$.

This means that a PLOFF φ is equivalent to an energy function E if the violation rank of φ is equal to the characteristic function of E, i.e.

$(\forall x)(Vrank_\varphi(x) = Erank_E(x) + c)$.

The following example shows the equivalence between a PLOFF and the energy function it represents.

Example 5.3 (continued)

The PLOFF can be written in the form:

$\varphi = \{<3000,T>, <1000,T\rightarrow H>, <1000,T\rightarrow R>, <10,R\rightarrow\neg W>, <10,H\rightarrow W>\}$.

Each of the pairs is converted to an energy function, which will be a term in the complete energy function, which in turn computes the characteristic of the negation of every sub-formula φ_i. For example the negation of the sub-formula $\neg T\vee H$ (i.e. $T\rightarrow H$) is $\neg(\neg T\vee H) = T\wedge\neg H$. Its characteristic can be written as $T(1-H) = T-TH$.

The energy functions associated with the sub-formulas are:

$3000(E_T) = 3000(-T)$

$1000(E_{\neg T\vee H}) = 1000(T-TH)$

$1000(E_{\neg T\vee R}) = 1000(T-TR)$

$10(E_{\neg R\vee\neg W}) = 10(RW)$

$10(E_{\neg H\vee W}) = 10(H-HW)$

Adding the energy terms, we have: $E = -1000T-1000TH-1000TR+10RW+10H-10HW$.

The negative terms correspond to positive weights and the positive terms correspond to negative weights in the network. This is so because the procedure to obtain the energy function includes the computation of the characteristic of the negation of each sub-formula. The corresponding network is shown in the following figure:

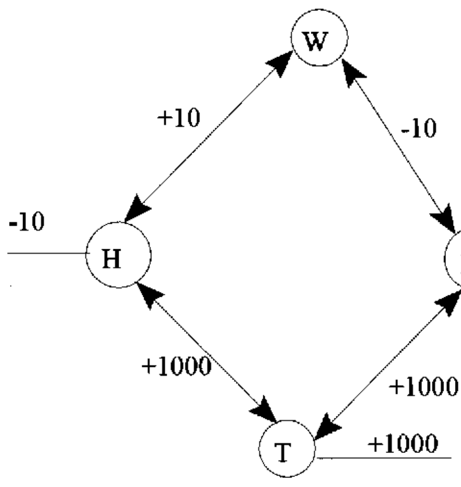


Figure 5.3 ■

The network performs a search (which will be discussed in the next section) for a preferred model of φ, and hence a search for a preferred theory of φ. Inferencing in a neural network is viewed as an adaptive process, whereby weights and the activations of the nodes are updated according to the learning algorithm. This learning algorithm is repeated until some stable condition is reached (in which an optimum solution is reached). In a similar way the inference involved in Default logic systems may be viewed as an adaptive process [21] which searches for the reasonable belief in going through a conflict resolution process. This process was discussed in chapters 2 and 3. The link between inferencing in Default logic systems and inferencing in neural networks will be pursued further in section 5.4.

We have shown that the extended logic, namely *penalty logic*, makes it possible to represent nonmonotonic knowledge in neural networks. Penalty logic is also able to cope with inconsistency in the knowledge base. The equivalence between penalty logic, the corresponding energy function E and the symmetric networks has been shown.

We now describe an arbitrary connectionist network for simulating Default logic.

5.4 Simulating Default logic in neural networks

Given the discussion of default reasoning and related work in nonmonotonic reasoning, there is no logical reasoning system that is actually brave enough to discard axioms or cancel defaults in order to restore consistency to the knowledge base. In [21] Narazaki and Ralescu introduce an approach that accomodates this notion of cancellation. They propose a connectionist default inference method. This method consists of three steps:

1. Detection of a conflict.
2. Identification and ranking of competing defaults as is done in PLOFF.
3. Deciding which defaults to cancel, based on the preference theory for resolving conflict.

Before we consider how this inference mechanism is handled in the neural network proposed in [21], we will first see how defaults are represented in neural networks.

5.4.1 Knowledge representation of defaults in neural networks

Consider the default rule A:B/C. Assume A, B, and C are disjoint in that they do not have any common propositions. The rule can be interpreted as :

$\neg A \vee (B \wedge C) \vee (\neg B \wedge \neg C)$, showing that

1. If A is true, then either both B and C are true or both B and C are false
2. If A is false, it does not matter whether B and C are true or false.

This can be illustrated by the example that follows.

Example 5.5

Summer: ¬Rain/Sun_Shining. If it is true that it is Summer then either it is not raining and the sun is shining or it is raining and the sun is not shining. If it is not Summer then, for the purposes of this rule, it really does not matter whether it is raining or whether the sun is shining. ◘

Expanding the above interpretation, yields the following two implications:

$A\wedge B \rightarrow C,\ A\wedge C \rightarrow B$ ①

Note that $A\wedge B\rightarrow C$ is equivalent to $\neg(A\wedge B)\vee C = \neg A\vee\neg B\vee C$ and $A\wedge C\rightarrow B$ is equivalent to $\neg A\vee\neg C\vee B$. Hence we have $(\neg A\vee\neg B\vee C) \wedge (\neg A\vee\neg C\vee B) = \neg A \vee(\neg B\wedge\neg C)\vee (B\wedge C)$, which is equivalent to the original interpretation.

Using the translation rule in ① we can do the same for a normal default rule and a seminormal default rule. Applying the translation rule to the normal default rule A:B/B yields the implication $A\wedge B \rightarrow B$, and applying the rule to the seminormal default rule $A{:}B\wedge C/C$ yields the implications $A\wedge B\wedge C \rightarrow C$, and $A\wedge C \rightarrow B\wedge C$. The implications obtained from normal and seminormal defaults contain a tautology, that is, a wff that is true under every possible assignment of truth values. Generally our *thinking process* is concerned with resolving possible conflicts caused by default knowledge and the method of cancelling defaults based on our preference is used to resolve the conflicts. Quantifiers such as, 'most', 'all', 'at least' and 'usually', can be used as clues for our preference. However, the tautology obtained using ① does not express a preference (i.e. believe q as far as situations allow). The absence of quantifiers make it difficult to determine a preference. In [21] Narazaki and Ralescu cope with this problem by introducing a *meta variable* f - a variable that is not actually part of the reasoning mechanism, but is used to aid the reasoning process. The meta variable is used to get rid of the tautology. The justification B is replaced by the meta variable f. The normal default A:B/B is now changed to A:f/B, and the translation rule in ① is applied to yield

$A \wedge f \rightarrow B$ and $A \wedge B \rightarrow f$. ②

In the same manner the seminormal default $A{:}B \wedge C/C$ is modified to yield $A{:}B \wedge f/C$. Applying the translation rule in ① yields the following implications:

$A \wedge B \wedge f \rightarrow C$ and $A \wedge C \rightarrow B \wedge f$. ③

We have $(A \wedge B \wedge f \rightarrow C) \wedge (A \wedge C \rightarrow B \wedge f) = \neg A \vee (\neg B \wedge \neg C) \vee (\neg f \wedge \neg C) \vee (C \wedge f \wedge B)$. In other words f divides the beliefs into two parts.

Now that we are able to represent the default rules in the form of wffs using meta variables, defaut rules can be mapped into a neural network using the approach of Penalty logic.

5.4.2 Connectionist inference mechanism

The network inference method proposed in [21] is based on the concept of simultaneous constraint satisfaction. Based on the Hopfield neural network, which is generally suited to solve optimization problems, the knowledge is translated into constraints and the energy function is defined as the sum of the constraint violations. The constraints correspond to the conflicting propositions. In order to reduce the extent of constraint violations a *relaxation* process is employed. This process updates the truth value distribution to decrease the energy, and to restore a more stable state. You may refer to the discussion of this in chapter 4. A detail account of the inference method follows.

The inference mechanism consists of two processes: the *network* mechanism and the *logical* mechanism. The network mechanism implements a relaxation method, which will be discussed later, and the logical mechanism resolves any conflict that might occur based on the network state. The nodes of the network are classified into three kinds of nodes, namely

1. constraint nodes, which represent constraint equations,
2. proposition nodes, which represent facts, and
3. one global node which controls the solution.

When we speak of a problem, we mean finding (i.e. deriving) a conclusion amidst the axioms and default knowledge available, or finding the 'best' solution, as in the classical Travelling Salesman Problem. The solution obtained would be interpreted from a graphical perspective. Hence the solution can be a local solution or a global solution. A local solution is when we find one rule of which the premise is satisfied, we assert that the consequent is true without taking into consideration the effect of other rules. A global solution is a macroscopic solution where the entire knowledge base or network is considered. We will refer to the global solution as a *feasible* solution. When a feasible solution is reached, the truth value distribution makes the energy 0. This would correspond to a consistent belief. However, if the energy attains a positive minimum value it is said to have a local solution, which is not satisfactory. The local solution is not indicative of the interactions of all other rules available.

The relaxation process searches for a solution. If a local solution is achieved, then the value of the *global* node goes to infinity, and when an upper limit (user-defined) of the global node is exceeded, then a kind of confusion is triggered which allows the network to get out of the local solution. A snapshot (the value of each node and the weight on each of the arcs) of the state of the network when a solution (feasible or local) is reached, is sent to the *logical* mechanism. The logical mechanism performs the task of controlling the search process based on the snapshot of the network it received. Because of the uncertainty (inconsistency) of the knowledge base, the truth values are allowed to be fuzzy or continuous in the range [0,1]. The truth value is updated based on the gradient information of a proposition. The gradient information of a proposition x is given by

$\phi(x) = \sum_{j=1}^{n} p_j\, e_j\, (\delta e_j / \delta x)$, where p_j is the weight (penalty) and e_j is a constraint

violation of the jth piece of knowledge. For a fact whose truth value is fixed, the gradient is zero. Many solutions to a problem are possible and we need to find a preferred solution. In order to find this preferred solution the relaxation process is iterated and each time the initial truth values are changed. It is, however, not possible to know when all solutions have been found. Also, if the relaxation process keeps finding local solutions, it is not possible to know whether a feasible solution really exists or whether more time is required to find this feasible solution. Therefore the time required to continue the search must be determined heuristically (using rules of thumb which are not necessarily true but are helpful guides to finding solutions). A preference criterion is required to order the solutions. The penalty logic developed earlier uses the minimality of the constraint violations as such a criterion. The preference criterion can be a function that maps a truth value distribution to a scalar value. The network mechanism concentrates on providing the logical mechanism with the network's state, whilst the logical mechanism orders and selects the best solution. Because the search time required is heuristic in nature, the network makes use of a motivation function m(t) to help determine the search time necessary. This function may be defined as $m(t) = e^{-\alpha t}$, where t indicates the number of times the relaxation process was applied since the current best solution was obtained, and $\alpha > 0$ is a parameter that controls the waiting time. If the best solution is a feasible solution, $m(t) \rightarrow 0$ indicates that the relaxation process converges to a model or preferred theory. The built-in program decodes the truth values of the proposition and constraint nodes to a user-understandable interpretation and hence a solution is found.

A conceptual diagram [22] of the inference mechanism follows.

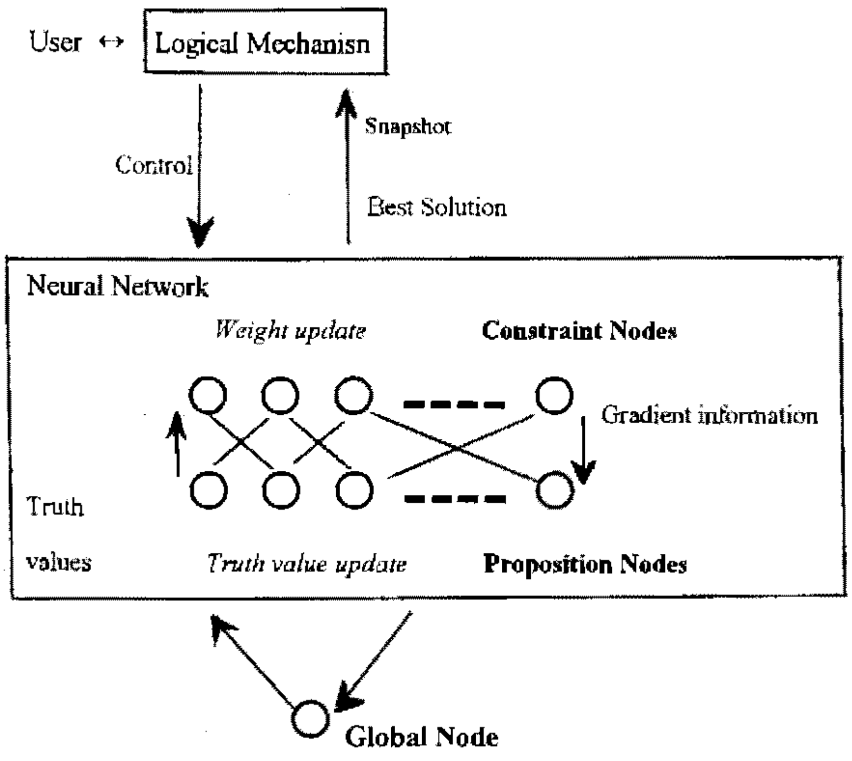


Figure 5.4

Constraint nodes:	These nodes correspond to the constraints expressed by rules and general constraints. They are connected to the related proposition nodes. The built-in program calculates the amount of constraint violation.
Propositional nodes:	These nodes are connected to the related constraint nodes. The built-in program updates the truth value associated with a proposition node.
Global node:	This node is connected to all the constraint and proposition nodes to transmit global information like the maximum constraint violation.

Next we need a strategy to resolve conflict. Narazaki and Ralescu discuss this strategy, using meta-relaxation. This is the topic of the next section.

5.4.3 Conflict resolution strategies using meta-relaxation

If there are *n* defaults then there will be *n* meta variables in the belief set. We recall from chapter 2 that the applicability of a default depends on whether the prerequisite is true and whether the justification is consistent with the belief set. The translation of the defaults to wffs using meta variables as described in paragraph 5.4.1, indicates that the meta variable replaces the justification (or part thereof). Therefore the justification which is now the meta variable must be true. Hence the new form of the defaults written as wffs require the meta variable to be true in order that the defaults be applicable. Another constraint equation indicating the maximum number of propositions that can be completely true, is given in the form $\sum_{i=1}^{n} V(f_i) = l$, where $V(f_i)$ denotes the truth value of meta variables f_i in $\{0,1\}$. In this approach the truth values are allowed to be fuzzy, i.e. somewhere in $[0,1]$. However it is possible to keep to truth values in $\{0,1\}$ using an integer programming approach. l will denote the sum of the truth values of f_i, thus showing the number of applicable defaults (f_i is the meta variable used in the default). A process to decrease the number (l) of completely true defaults is required, so that the possibility of conflict(s) in the current belief set may be reduced. This process is a meta-relaxation process which is iterated until the conflict is resolved. Before the meta-relaxation is performed, a relaxation process, called *object level relaxation* is required. This relaxation process updates truth values of propositions under the given value of l. Only those defaults that correspond to the meta variables in the conflict set are considered. If a solution is not likely to appear (i.e solution is local), it requires the motivation function and the meta-relaxation is implemented to decrease l. Then the object level relaxation is performed again, starting with a decreased l. This process of object level relaxation and meta-relaxation is iterated until a feasible solution is found. Note that the relaxation process is really the learning algorithm required by the network. The key idea is to minimize the number of cancellations. Remember the reason for cancellations is to resolve conflicts. The meta-relaxation is performed by the *logical* mechanism. The three steps involved are:

1. A decision is made regarding the existence of a feasible solution. This is done via

the motivation function *m(t)*, discussed earlier.

2. If it is not possible to find a feasible solution then a conflict set is extracted from the best local solution. At this point it is not known which defaults are causing the conflicts. A set such as $F = \{f_k;\ z_k > \epsilon_0\}$ is called a conflict set, where z_k is a constraint violation for the default associated with f_k (the meta variable used in the justification of the default) and ϵ_0 is a threshold that is user-defined. This set contains all the meta variables f_k of the local best solution.

3. Defaults corresponding to f_k in the set F are then removed to resolve the conflict. In other words the meta variable f_k corresponding to a default is removed from the conflict set F. This is done in two steps: setting $V(f_i) = 0$ and decrementing *l* by one.

There are a number of cancellation strategies that Narazaki and Ralescu propose. I will not discuss these strategies in this dissertation. For details the reader can refer to [21]. The strategy used is dependent on the type of application.

Let us illustrate the use of defaults in a neural network by an example taken from [21].

Example 5.6

Consider the following rules (denoted by R_i, i = 1,..., 7) and their corresponding defaults (R_1,..., R_5) and axioms (R_6 and R_7).

R1: Usually things do not fly.	$\dfrac{\text{Thing}(x){:}\neg\text{Fly}(x)}{\neg\text{Fly}(x)}$
R2: Usually birds fly.	$\dfrac{\text{Bird}(x) : \text{Fly}(x)}{\text{Fly}(x)}$
R3: Penguins don't fly.	$\dfrac{\text{Penguin}(x){:}\neg\text{Fly}(x)}{\neg\text{Fly}(x)}$
R4: Caddy seems to be a Penguin.	$\dfrac{\text{Caddy}(x){:}\text{Penguin}(x)}{\text{Penguin}(x)}$

R5: Birds usually eat insects. $\dfrac{\text{Bird}(x):\text{Eat\_Insects}(x)}{\text{Eat\_Insects}(x)}$

R6: Birds are things. $\text{Bird}(x) \rightarrow \text{Thing}(x)$

R7: Penguins are birds. $\text{Penguin}(x) \rightarrow \text{Bird}(x)$

Applying the translation rule in ① (page 82) to the defaults and replacing the justification part of the default rule by meta variables, yield the following implications:

R1: $\text{Thing}(x) \wedge f_1 \rightarrow \neg\text{Fly}(x)$, $\text{Thing}(x) \wedge \neg \mathit{Fly}(x) \rightarrow f_1$

R2: $\text{Bird}(x) \wedge f_2 \rightarrow \text{Fly}(x)$, $\text{Bird}(x) \wedge \text{Fly}(x) \rightarrow f_2$

R3: $\text{Penguin}(x) \wedge f_3 \rightarrow \neg\text{Fly}(x)$, $\text{Penguin}(x) \wedge \neg\text{Fly}(x) \rightarrow f_3$

R4: $\text{Caddy}(x) \wedge f_4 \rightarrow \text{Penguin}(x)$, $\text{Caddy}(x) \wedge \text{Penguin}(x) \rightarrow f_4$

R5: $\text{Bird}(x) \wedge f_5 \rightarrow \text{Eat\_Insect}(x)$, $\text{Bird}(x) \wedge \text{Eat\_Insect}(x) \rightarrow f_5$

R6: $\text{Bird}(x) \rightarrow \text{Thing}(x)$

R7: $\text{Penguin}(x) \rightarrow \text{Bird}(x)$

We consider R6 and R7 to be facts.

We assign a truth value of 1 to Caddy, i.e. $V(\text{Caddy}) = 1$ and we set $V(f_i) = 1$ for i= 1,2,..,5. All the defaults are used. In this case there would be no feasible solution because all five defaults cannot be true at the same time, although the individual statements make sense. (All the defaults are part of the conflict set F). A conflict occurs in determining whether Caddy flies or not. According to R4 Caddy is a penguin, and R3 states that penguins do not fly and at the same time we know that penguins are birds and by R2 birds should fly. We can either choose R3 and therefore say that Caddy cannot fly and cancel R2, i.e. birds fly, or we may doubt that Caddy is actually a penguin, cancelling R4 and in the process conclude that Caddy flies. From this we notice that we make a choice depending on our preference. Also the person who is not familiar with penguins would prefer cancelling R3 and R4, i.e. penguins do not fly. This person may never cancel R3 again if convinced that penguins do not fly. This indicates that a preference is a result of a learning mechanism. ◘

Since there is no feasible solution because *l = 5*, a meta-relaxation is performed. Narazaki and Ralescu propose a number of strategies that can be used in determining which defaults should be cancelled. Some of the strategies they use are *Free Strategy, Preference Oracle, Specificity Strategy* and *Object Level Strategy*. In this dissertation we will discuss one of these, namely object level strategy.

Obiect level strategy

We start this discussion by defining the specificity index of a proposition.

Definition 5.6

The specificity index of a proposition is given by $S(p) = | V(p) - 0.5 |$, where $V(p)$ indicates the truth value of p.

The specificity index is calculated for all propositions having variable truth values as in R1 to R5 in example 5.6 above. R6 and R7 are facts therefore their truth values are fixed at one. The proposition (p) that has the highest specificity index after each relaxation process is identified and then the truth value of p is set. The process of implementing the relaxation process, the identification of the proposition with the highest specificity index and setting the truth value of the proposition, is iterated. This iterative procedure results in reducing the conflict set F. The defaults that correspond to the remaining meta variables f_k in F are cancelled. In this case Penguin(x), Fly(x), Thing(x), Eat_Insect(x) and Bird(x) are in turn fixed to either 0 or 1, so these values are not fuzzy. The conflict set F eventually has only one default, i.e. "Birds fly" and therefore the cancellation of R2 is in order.

We will discuss another example and see how the inference method proposed in [21] can deal with interacting pieces of knowledge.

Example 5.7 (taken from [21])

Consider a basic air-conditioner (AC) problem. We assume that the AC is able to heat and cool the air in the room by selecting the appropriate mode, ie. Heat mode (HM) and Cool mode (CM). The AC operates according to the following rules:

1. If AC is switched on and the HM is turned on then *warm* air will be emitted.
2. If AC is switched on and the CM is turned on then *cool* air will be emitted.
3. The AC is turned on with either the "cool mode"on or the "heat mode" on, i.e. CM$\vee$ HM.

We now transform the above knowledge into defaults and wff(s) as follows:

AC: $warm \wedge \neg$CM/$warm$, AC: $cool \wedge \neg$ HM/$cool$, CM$\vee$ HM.

It may be erroneously concluded that both warm and cool air are emitted if the interactions among defaults are not properly dealt with. We observe that if the first default is used, its justification $\neg$CM should block the application of the second default because of the wff CM$\vee$HM. To prevent the erroneous conclusion that "warm and cool" air come out, the defaults are translated using meta variables f_1 and f_2 as follows:

AC$\wedge f_1 \wedge \neg$CM $\rightarrow$ $warm$, AC$\wedge$ $warm \rightarrow f_1 \wedge \neg$CM

AC$\wedge f_2 \wedge \neg$HM $\rightarrow$ $cool$, AC$\wedge$ $cool \rightarrow f_2 \wedge \neg$ HM

According to the inference mechanism, f_1 and f_2 are set to 1 (true). The object level relaxation yields one of the following possible solutions: $warm \wedge \neg\ cool$, $\neg warm \wedge\ cool$, or $\neg warm \wedge \neg cool$. The last solution can be avoided by replacing the wff CM$\vee$HM with CM Xor HM where Xor denotes an "exclusive or". ◘

Many types of reasoning can be viewed in terms of *constraint satisfaction problems* (CSP). Examples of such reasoning are temporal reasoning [4], truth maintenance [6] and diagnostic reasoning [5]. We have seen that constraint satisfaction problems can be modelled in neural networks. Default reasoning which is closely linked to diagnostic reasoning can also be viewed in terms of CSP. From all that has been discussed so far it would seem that if β is in some extension of the default theory involved then β would be one of the solutions obtained in a neural network. In chapter 3, paragraph 3.3 we discussed the undecidability of the proof theory of Default logic. In contrast, the neural network approach will always yield a solution if it exists.

5.5 Conclusion

Penalty logic has been developed by Pinkas [23] as a platform to represent nonmonotonic knowledge. Pinkas' theory involves capturing the framework of nonmonotonic logic in a neural network, using penalty logic. It has been shown that for every Penalty logic formula there is an equivalent energy function, which can be represented in a symmetric connectionist network.

Narazaki and Ralescu [21] show a way of representing Default logic in a neural network. Their approach is one in which default rules are translated into implications which are in turn converted to formulas of ordinary logic. Just as ordinary logic can be mapped into a neural network so too defaults (with the necessary conversion) can be represented in a neural network. In the case of resolving conflict, defaults are cancelled based on a preference, which is built into the neural network.

This brings us to the completion of this final chapter. In this dissertation we have provided an exposition of Default logic together with a proof theory for Default logic. We have presented a proof theory for Default logic based on prioritised conflict resolution and it has been shown that Default logic can be mapped in a neural network. In order to deal with the inference in a conflicting situation, the inference problem was shifted from the logical problem to the optimisation problem, in which maximum consistency is aimed at. The inference mechanism presented is an adaptive inference that is capable of cancelling defaults in order to resolve conflict(s). It has been found that the continuous Hopfield network is appropriate in modelling default reasoning. The power of the Hopfield network lies in its parallelism. In addition to this parallelism, a relaxation method was presented that can cope efficiently with the interactions among knowledge. A typical optimisation problem, the TSP, has been presented to demonstrate the theory. The selection between the possible answers it might give, shows the power of the computation a neural network is capable of. The proposed mechanism simulates an "intuitive" part of the thinking process. However, further research is required to investigate a "conscious" part of the thinking process, which is referred to as the logical mechanism that works on snapshots of the network state. This might aid in the implementation of complex default reasoning in neural networks.

REFERENCES

[1] Besnard, P. 1989. *An introduction to Default logic*. Springer-Verlag.

[2] Brewka G. 1989. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of 11th International Joint Conference on Artificial Intelligence (IJCAI 89)*, pp. 1043-1048.

[3] Clark, K. Negation as Failure. In Gallaire, H and Minker J. (Eds), *Logic And Data Bases.* New York: Plenum Press. pp. 55-76.

[4] Dechter, Meiri and Pearl 1989. Temporal; constraint networks. In Brachman, Levesque and Reiter (eds.) *Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference. San Mateo,* ca: Morgan Kaufmann.

[5] Dechter and Pearl 1988. Constraint-directed approach to diagnosis. *UCLA CS Dept. Tech. Rept.R-72-I.*

[6] de Kleer, J. 1989. A comparison of ATMS and CSP techniques. *Proceedings of the eleventh International Joint Conference on Artificial Intelligence,* pp. 290-296.

[7] Dinsmore, J. 1992. *The Symbolic and Connectionist Paradigms, Closing the Gap.* Lawrence Erlbaum Ass., Publishers.

[8] Doyle, J. 1981. A truth maintenance system. *Artificial Intelligence 12,* pp. 231- 272.

[9] Etherington David, W. 1987. A Semantics for Default logic. *Proc.Tenth International Joint conference on Artificial Intelligence,* Aug 1987.

[10] Fausett, L. 1994. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice Hall, ISBN 0-13-042250-1.

[11] Gärdenfors, P. 1991. Symbolic and Quantitative Approaches to Uncertainty. *ECSQAU Marseille, France*. October 1991.

[12] Genesereth, M.R. and Nilsson, N.J. 1987. *Logical Foundations of artificial intelligence*. Morgan Kaufman.

[13] Hopfield, J.J. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences 81*, pp. 3088-3092.

[14] Hopfield, J.J. and Tank, D. W. 1995. "Neural" Computation in Optimization Problems. *Biol. Cybern., vol. 52*, pp. 141-152.

[15] Junker U. and Brewka G. 1991. Handling partially ordered defaults in TMS. *European Conference ECSQAU'91, vol. 548 of Lecture Notes in Computer Science*. pp. 211-218.

[16] Lukaszewicz, W. 1985. Two Results in Default logic. *Proc. Ninth International Joint conference on Artificial Intelligence, Aug 1985*, pp. 459-461.

[17] Lukaszewic, W. 1990. *Nonmonotonic Reasoning - Formalization of commonsense reasoning*. Chapter 5, pp. 159-224. Ellis Horwood.

[18] McCarthy, J. 1980. Circumscription-A Form of Nonmonotonic Reasoning. *Artificial Intelligence, 13*. pp. 27-39.

[19] Mengin, J. 1994. A theorem prover for Default logic based on prioritized conflict resolution and an extended resolution principle. *Lecture notes in AI vol. 946*.

[20] Mengin, J. 1994. Prioritized Conflict Resolution for Default Reasoning. *ECAI 94. 11th European Conference on Artificial Intelligence*. pp. 376-380.

[21] Narazaki, H and Ralescu A. L. 1993. A Connectionist approach for default inference and conflict resolution. *IEEE New York, USA*.

[22] Narazaki, H and Ralescu, A.L. 1992. A Connectionist Approach for Rule-Based Inference Using an Improved Relaxation Method. *IEEE Transactions on Neural Networks, Vol. 3 no. 5*, pp. 741-751.

[23] Pinkas, G.1994. Propositional Logic, Nonmonotonic Reasoning and Symmetric Networks- On Briging the Gap Between Symblic and Connectionist knowledge Representation. *In Neural Networks for Knowledge Representation and Inference.* Lawrence Erlbaum Ass., Publishers.

[24] Reiter, R. 1978. On Closed World Data Bases. In Gallaire, H and Minker J. (Eds), *Logic And Data Bases.* New York: Plenum Press. pp. 55-76.

[25] Reiter, R., Criscuolo, G. 1983. Some representational issues in default reasoning. *International Journal of Computers and Mathematics, 9(1):* pp. 15-27.

[26] Reiter, R. 1980. A Logic for Default Reasoning. *Artificial Intelligence 13*, pp 81-132.

[27] Rosenblatt, H. 1993. Default logic and Diagnostic Reasoning. *Msc Thesis, UNISA* .

[28] Shastri, L.1990. Connectionism and the computational effectiveness of reasoning. *Theoretical Linguistics 16:1*. pp. 65-87.