# Aspects of the Inheritance Problem

by
Andrew Cripps
B.Sc., University of Reading, 1986

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

ACCEPTED

in the Department
of
Computer Science

We accept this dissertation as conforming
to the required standard

Dr. H.A. Müller

Dr. N. Cercone

Dr. J. Foss

Dr. W. Wadge

© Andrew Cripps, 1987
UNIVERSITY OF VICTORIA
March 1987

*All rights reserved. This dissertation may not be reproduced in whole or in part, by mimeograph or other means, without the permission of the author.*

## ABSTRACT

Trying to represent knowledge in computers has long been an established field of research in artificial intelligence. The problem of inheritance is located within the field of knowledge representation. This thesis provides a general survey of the problem, exploring its history and development, for the first time. The first part describes the evolution of the problem and offers an account of knowledge-based systems which have made use of inheritance. The second part of this thesis shows that these systems are shown to have several shortcomings, and by identifying them, the problem of inheritance is carefully specified. The last part of the thesis provides and efficient solution to a sub-problem of the inheritance problem, called the problem of exceptions.

[redacted]

Dr. H.A. Müller Dr. N. Cercone

[redacted]

Dr. J. Foss Dr. W. Wadge

# TABLE OF CONTENTS

# LIST OF FIGURES

# ACKNOWLEDGEMENTS

My thanks to Nick Cercone and Carol Murchison for their valuable assistance and advice. I should like to acknowledge all the other graduate students without whom this work would have been completed much more quickly.

# CHAPTER 1

# Introduction

*"Damnosa hereditas"*
*— Gaius,*
*Institutes, ii, 183.*

## 1.1. Introduction

In beginning a long piece of work such as a Masters thesis, one is inevitably caused to think in new ways about old problems. There is both a comfort and a pleasure in tackling a problem which has vexed many influential researchers in the field, finding that one can add at least something to the body of knowledge about that problem. The problems of inheritance have been recognised, though not stated formally, from the very first in knowledge representation.

In a little over two decades, we have developed computers which seem to exhibit something like human intelligence. Unfortunately however, it is all too often a façade which crumbles outside a narrow domain of discourse. The problems of representing knowledge in a way which will enable computers to reason as do humans are not solved. Inheritance is one such problem. In this thesis the background to the problem is explained and all the threads of research about the

subject are drawn together in order to present a thorough account of the problem. Some problems with the most advanced solutions are pointed out and improvements to them are suggested. This chapter is an introduction to the issues at hand.

## 1.2. The general area of research

### 1.2.1. The field of knowledge representation

Knowledge representation has become a well defined research area within artificial intelligence. One of the goals of this research is to find ways of representing knowledge about the real world in computers. If this was the only goal, then it would have already been achieved. The telling condition which has thus far been omitted is that the system which exploits the knowledge held within the computer must be able to behave in a similar manner to a human with the same knowledge. It is a surprising fact that humans can answer questions making reference to highly complex concepts very easily. For example, if you are asked to imagine a heavy fall of snow, you instantly know that it would be inappropriate to go out in your shorts; you know something about the *weather*. Not only so, but you know what sort of clothing would be appropriate, you know that it is not a good day for the beach or for cricket. More surprising still is one's ability to know that one does not know something. If asked, "Do you know when the dirigible *Hindenburg* went down, killing thirty-six people?", you

would instantly supply the answer or reply in the negative.[1] Observing that one is able to answer so quickly belies the fact that what is going on in the brain is not some naïve linear search through an enormous number of facts; such a search would not be possible in so short a time. Ideally, we would like the computer to work with such apparent effortlessness. Unfortunately, we have not come close to building a computer system which handles knowledge as effectively as do humans. To find some adequate representation for knowledge is the broadest possible specification of the problem which concerns the author.

### 1.2.2. The notion of an inheritance system

Three methods of representing knowledge have become prevalent in the field: (a) semantic networks; (b) frame based systems; (c) predicate calculus. Both semantic networks and frame-based knowledge representation systems employ inheritance. An inheritance system is a knowledge representation system founded on the hierarchical structuring of knowledge. Almost all knowledge representation languages and object-oriented programming languages have been organised around such systems. Semantic networks and frames embody inheritance in the form of a hierarchical ordering of objects. Two systems stand out as landmarks in the development of inheritance: FRL [GoR77, RoG77] and NETL [Fah79].

---

[1] The correct answer is May 6, 1937.

The hierarchies used in these systems are implemented using links which show inheritance. For example, "*A dog IS-A mammal*," "*Lapis lazuli IS-A semi-precious gem*," "*Jemimah IS-A cat*". The statements most often handled by knowledge representation systems are *predication* and *universally quantified conditionals*. Predication simply asserts that an individual is of a certain type or class, perhaps *pilot* or *male*. The latter statements say such things as: "If X is a Y then X is also a Z." Thus, given the universal statement, "All canaries are yellow," if Joey is a canary, then Joey is yellow. Using a hierarchy built up from IS-A links, it is easy to distribute inheritable properties throughout a semantic network. Properties which can be shared by many individuals are placed as high up the hierarchy as possible, to maximise the number of subclasses which may inherit them, since attributes are inherited by all nodes below the one in which the attribute is stored.

Inheritance may be thought of as *modus ponens* being used over and over again [EtR83]. *Modus ponens* is a hypothetical form of syllogism in which there is a hypothetical major premise and a categorical minor. For example,

If $p$ then $q$
$p$
---
$q$

The more readable form of *modus ponens* is:

If A is B then C is D.
A is B

C is D

When, as above, the antecedent of the hypothetical major premise is asserted, the argument is said to be constructive and in the *modus ponens*. As an example of affirming the antecedent, consider this:-

If a submarine is yellow, then it has been painted.
The *Beetle* is yellow.
∴ It has been painted.

A defining characteristic of inheritance systems is that they use a hierarchical ordering of classes to control inference implicitly; if there are exceptions to generalisations in these systems, they are never listed explicitly. Systems such as FRL and NETL do not handle exceptions well. It is important to handle all cases correctly, so as not draw invalid conclusions, but neither NETL nor NUDGE[2] do this.

### 1.2.3. Links

Exceptions complicate inheritance systems. Without such complications, an inheritance system is a simple taxonomy organised by links. The interconnecting links of the taxonomy are usually called *IS-A* links because all the relationships are of the form, *X IS-A Y*[3]. Representing both individuals and 'concepts' in the

[2] NUDGE was the prototype system for FRL.

[3] IS-A is the most common form for a group of links including A-Kind-Of, Part-Of and Member.

network in this way makes inheritance a natural property of the representation. Figure 1.1 below is an example of a basic taxonomy. As the previous section demonstrated, property inheritance may be seen as the repeated application of *modus ponens* [EtR83]. With simple taxonomies, it is not so much that we have chosen to ignore exceptions but rather that we must exclude them. If X is a subclass of Y, then X is necessarily a subclass of Y, and must inherit its properties; it cannot be an exception.

Semantic networks were first used to represent concepts and became increasingly popular as a way of representing a wide range of objects. The idea of semantic networks became common knowledge with the publication of Quillian's

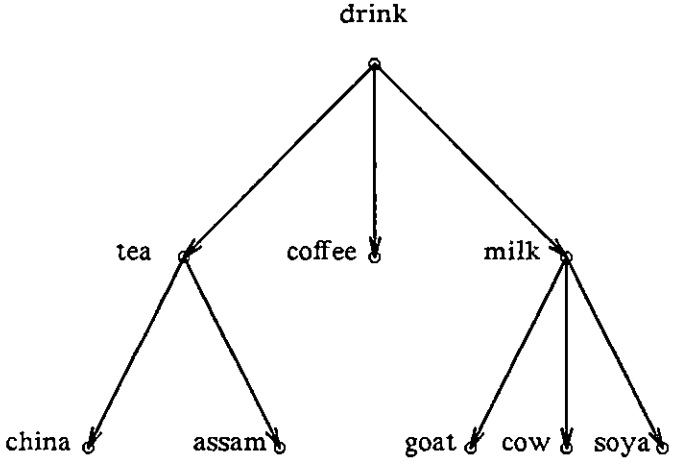


**Figure 1.1 A basic taxonomy**

Ph.D. thesis in 1966 [Qui66]. Quillian's aim was quite straightforward. He wanted to represent words and their meanings in his networks. This narrow use was soon lost, and the term 'semantic network' was applied to all sorts of data structures, appearing *hic et ubique*. At this stage, a great deal of confusion arose as to exactly what a semantic network is. Brachman [Bra79] tried to discover the essence of a semantic network and attempted to explain why a lot of early networks were deficient.

Brachman identified five levels of knowledge, each having some semantic content, on which all kinds of systems have been developed. Only one of these levels is of particular interest here. Of the five, Brachman claims that it is the epistemological level which has been the most misunderstood, but which has the greatest potential. The epistemological level is the one which deals with such topics as inheritance, abstraction, and the structuring of concepts. At the time that Brachman published his paper, new systems such as FRL [RoG77] and KRL [BoW77] were developed at this level.

Quillian's semantic networks are generally acknowledged to be the prototypes for those which are prevalent today. His representation consisted of nodes and interconnecting links to show associativity. Each node represented a word concept, and links radiated from each node to related nodes. These nodes and interconnecting links formed a dictionary.

This early work is interesting because it shows how we arrived at semantic networks and how inheritance came to light as a property of these simple taxonomic networks. Quillian made an important blend from the superclass-subclass taxonomy and the attribute/value description of properties for each class. He later revised this union and proposed the Teachable Language Comprehender (TLC) in 1969 [Qui69]. Although this system never reached its optimistic objectives, it did point the way to the first real work on inheritance. Collins and Quillian [CoQ69] conducted a series of experiments to determine whether the TLC was a plausible model for human memory. The networks in TLC were simple hierarchies of concepts, such as person, fish, animal, and car. Attached to each node was a set of properties which defined the corresponding concept. Properties common to many concepts were placed as high up in the hierarchy as possible. It would therefore be expected that specific properties local to a very few concept nodes would be verified by the system more quickly than properties of a more general nature. For example, very high up in the mammal hierarchy would come properties such as ". . . is viviparous," ". . . is warm-blooded." Further down the hierarchy would be properties such as ". . . has trunk," specific to one or two mammals. It would be expected that TLC would assert the latter more quickly than the former. The results of this experiment lent weight to the idea that human memory is likewise organised hierarchically—but this was by no means a conclusive study. More importantly, property inheritance was understood to a greater degree. It also gave rise to a concrete notion of "semantic distance"

between concepts, that is to say, how many links must be traversed to get from one node to another.

Carbonell became interested in Quillian's work and was motivated to write SCHOLAR [BBB70, Car70]. This program contained a knowledge base describing the geography of South America. Carbonell noticed that he was representing two different kinds of item: concepts and instances. For example, the location of a place is a concept, a direction is a concept, but *Chile* is an instance of a country. This distinction paved the way for the development of the notion of instantiation.

All of the systems produced about this time overlooked one very important aspect of semantic networks concerning inheritance: there is a distinction between particular individuals (or concepts) and generic concepts. Because no distinction was made between these two, properties of individuals became confused with properties of generics. The individual node may be identified as an individual by means of a tag or simply by its name. The properties attached to that individual are contingent. With a generic node however, the properties themselves define what the node represents. Thus individual nodes may be taken as they stand without regard as to whether their properties adequately define them. Generic nodes however, if they represent concepts, rely upon their attached properties for their definition. This provokes a philosophical difficulty in that we cannot specify the defining properties of a thing. For example, we may think that whiteness is a *sine qua non* of snow. But one can imagine snow falling over a city which has a

severe smog cloud hanging over it which turns the snow brown. Would such smog-coloured snow actually be snow? If we want to say that it is snow, then whiteness is not one of its defining properties.

The distinction between general properties and specific, individual properties is vital for inheritance. Any inheritance mechanism must distinguish between generic properties and particular properties. Brachman and Schmolze produced a system called KL-ONE [BrS85], with which they hoped to address some of the problems regarding the epistemological status of semantic networks which Brachman discussed in his paper [Bra79]. The two main elements of KL-ONE were concepts and links. The concepts were allowed to be either generic or individual. Generic concepts were defined by their necessary conditions, and not by default assertions. Individual concepts represented individual objects and relationships by taking a more general concept and copying it, tailoring some of the properties of the general concept to make them specific to the individual. For example, Canterbury Cathedral is an "instance" of the *Cathedral* concept, and the individual concept of Canterbury Cathedral is emergent from the concept *Cathedral.* The individual concept denotes the real individual object. This individualisation represents a more formal and specific kind of property inheritance.

### 1.2.4. Frames

Another means of storing knowledge developed as the advantages of "chunking" knowledge were realised [Min75]. Storing knowledge as chunks

reduces the amount of time required to access the knowledge, since all the knowledge required is available after but one access. The first computational theory to make use of chunked knowledge was Minsky's frame proposal [Min75]. Frames soon became a research area in their own right. The important point about frames is a simple one: knowledge is grouped together into some concept or situation frame. Frames have been most useful for representing the epitome of a certain class of events or concepts.

For example, consider the frame of a hotel frame depicted in Figure 1.2 below. The empty slots in the frame must be filled when a particular instance of a hotel is recognised. Frame systems offer several mechanisms for filling in empty slots. The simplest method is to use default information wherever possible. For example, for the payment slot in Figure 1.2, we would at least be able to put "cash" in it, but not be able to say anything about the type of credit cards that are accepted by this hotel. The most complex form of slot filling is using inference mechanisms to enable a slot to inherit values.

There has been much debate as to when such inferences should be made. Some frames are highly procedurally oriented, leaving little room for making inferences. The debate about frame representations added to the continuing saga of the declarative/procedural controversy [Win75].

The essential differences between networks and frames stem from the observation that frames are largely functional rather than structural. A structure in

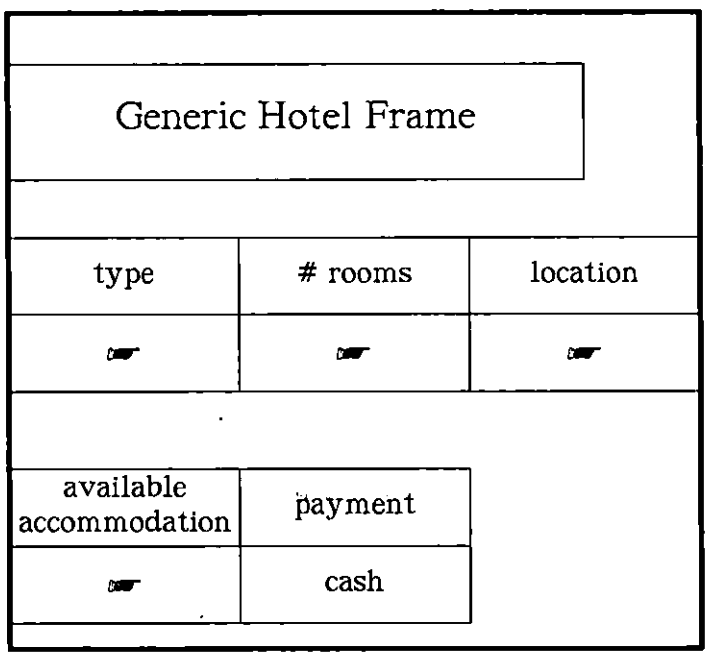


**Figure 1.2 A typical frame for a hotel room**

computer memory is called a frame because of the sort of knowledge it stores and not because it is radically different than any other kind of storage mechanism. To see how a particular slot might assume a value, consider the following example.

Suppose we want to build a gramophone disc frame. We might want to say that the cupboard has a collection of records in it, and the disc with the number *DSDL701* IS-A gramophone record. Properties of the disc will be the style of music, the pieces of music, the performers, the date of the recording, the colour of the disc, the material it is made of, the kind of sleeve. The difficulty arises when

only incomplete information about the disc is available. What should be done about the slots which remain unfilled?

First, we can use *default* values. If there is no specification of the colour of the disc, it is reasonable to assume that it is black by default. Secondly, it might also be convenient to activate a procedure when a value is placed in a slot. For example, if the diameter of the disc is less than twelve inches, we might want to activate a procedure to adjust the turntable in some way. Thirdly, we can make a procedure compute a needed value if there is neither a default value to be had nor an actual value available. If we consider frames to be nodes in a semantic network, then we may either incorporate procedures into frames where they are necessary to make inferences to fill slots, or after the slots have been filled by inferences, or inheritance may be employed. Inheritance comes into play when the frame slot remains empty despite the above techniques. It is possible to look for a value to fill the slot in frames related to the one given. The inheritance mechanism moves up in the *a-kind-of* hierarchy looking for an appropriate value in a slot in a frame at a higher level.

### 1.2.5. Hierarchies

In the real world, we conceive of things being ordered into some kind of hierarchy. This is particularly obvious in the animal and business worlds. An inheritance system is one which makes use of such taxonomies or hierarchies in computers. Recent examples of such systems with inheritance include KRL

[BoW77], FRL [GoR77, RoG77], and NETL [Fah79].

Suppose we want to represent creatures in our world which are grey, have four legs, large ears, and often live in fields. We might designate the owner of this collection of properties a "mouse." There is no reason why this abstraction (i.e. a collection of properties) should not be shared. For example, mice and elephants have some properties in common: both are viviparous and grey. Both of these animals could be included under another designator for a group of properties shared by both—perhaps mammals.

One tremendous advantage of hierarchical ordering is that it is an efficient method of representation. The creatures that we find in the world and wish to represent *could* all be represented with their own private set of properties. For computers this representation would be most inefficient. It would be more efficient if we had several mice to represent. In this case, we need only create a designator representing all the properties of mice that we wish to maintain, and include all the mice under it. Similarly, to efficiently store a whole range of animals, a higher class called mammals could be created which would have all the properties common to these animals.

Besides being a tidier way to represent knowledge about the world, hierarchies also make searching more efficient. If there was no ordering of the knowledge, the time to search the knowledge base would be excessive. The structure of the knowledge is also very important if we have more than one thing to

retrieve from the knowledge base. The ordering most suitable for one retrieval may be entirely unsuitable for another. For this reason, it is advantageous to have groupings in the hierarchy which overlap. This phenomenon is known as *multiple inheritance* (see below). Unfortunately, with the added flexibility come added problems from which simple inheritance systems do not suffer.

### 1.2.6. A brief introduction to the inheritance problem

#### Preliminaries

This section gives the reader something of the "flavour" of the problem. A more detailed account of the problem together with proposed solutions will be found in the next chapter. The simplest form of inheritance system is a straightforward hierarchy of inheritance. Knowledge stored in the hierarchy is made to conform to some rigid taxonomy. Hierarchical organisation is an integral part of our thinking and it is therefore natural to represent knowledge in computers according to a hierarchy which we already employ to describe the real world. Within this taxonomy, *IS-A* links show paths of inheritance. This hierarchical organisation makes it easy to distribute properties throughout a network chosen to represent it. Shared properties are placed as high up the hierarchy as possible. Attributes are inherited by all nodes below the one in which the attribute is stored, thus maximising the number of nodes which may inherit them.

Each node in the hierarchy represents either some particular individual such as "John," "Bonzo," or "Jessica," or it represents some particular concept such as

"telephone answering machine" or "battle." There are two principle kinds of statement that we want to represent in this simple network. The first is *predication*; we assert that a particular individual is a member of some class or type, or that some concept is a member of a super-concept. The following examples illustrate this concept.

> "*James is a boxer.*"
> or, expressed in first order predicate calculus form,
> boxer(james)
> "*A radio is a communications device.*"
> or expressed in logic form,
> communications-device(radio)

The second kind of statement most often represented is the *universally quantified conditional.* For example, "If James is a surgeon, then James uses a scalpel." This is the major premise of the syllogistic mood *modus ponens.*

### The problem of exceptions

There are two main aspects to the inheritance problem. The first issue is concerned with making provision for exceptions to generalisations. For example, a hierarchy may maintain that one property of "a piece of music" is that the piece is audible. Beneath the "piece of music" node, we find the works of composers such as Bach, Beethoven, and Honegger. How should some of the more experimental music which challenges the concept of art, such as John Cage's silent piece, *4'32"*, be classified? This piece is not audible, and is therefore an exception to the generalisation maintained higher up in the hierarchy. The most obvious solution

is to make an exception for this one piece to the general rule that music is audible at the node in the network which represents it, although an alternative solution with particular advantages will be presented in chapter four. Current reasoners, with simple inheritance schemes, such as FRL [GoR77, RoG77], will return that Cage's piece cannot be heard. Unfortunately however, because the search up through the inheritance hierarchy stops there, we lose the valuable generalisation that most music can be heard.

### Multiple inheritance

The second problem of inheritance is *multiple inheritance*. In a simple inheritance hierarchy, it is often the case that we wish to retrieve more than one piece of knowledge and each retrieval requires a different organisation of the hierarchy. Consequently, there will have to be multiple groupings of the properties in the hierarchy, and this is what is known as multiple inheritance.

Suppose that Spinoza is both a philosopher and a lens-grinder. Belonging to both classes he should rightfully inherit properties from both. But with a simple tree-like inheritance hierarchy this is not possible because it is only legal to inherit from one superclass. The only "efficient" way to represent Spinoza and his like-minded work colleagues would be either to create a "lens-grinding philosopher" class under an existing "philosopher" superclass, and restate all the properties of a lens-grinder, or create a "philosophising lens-grinder" class under "lens-grinder" and restate all the properties of the philosopher. Because we have

to repeat information, this "efficient" representation turns out to be inefficient. Another factor which makes it inefficient is that we must arbitrarily choose which class to create.

To escape from these difficulties, individuals must be allowed to inherit from multiple superclasses; there is no question of choice now as no new class need be created. Some systems permit multiple inheritance, such as Fahlman's NETL [Fah79]. Permitting multiple inheritance requires that the simple inheritance tree be abandoned in favour of a directed inheritance graph, where each node may have many superclasses. How this scheme complicates the semantics of an inheritance system will be shown in the next chapter.

### 1.3. A guide to the rest of the thesis

Chapter 2 gives an overview of previous systems and the way that they have tackled the issues of inheritance. These early systems all paved the way for an understanding of the issues; they are a product of research at a time when the problem of inheritance was not clearly specified or understood. Yet from these systems came an understanding of the basic problems of cancellation and exceptions.

Chapter 3 is an examination of the underlying concept of inheritance hierarchies. Brachman's work [Bra77, Bra79, Bra83] is extensively referred to since he has done much research in this area.

Chapter 4 provides a solution to the problem of exceptions and reveals certain deficiencies with Touretzky's work [Tou86].

Chapter 5 provides a summary of the thesis and draws some conclusions.

# CHAPTER 2

## The Problem and the Past

### 2.1. Introduction

The last chapter brought us to the point at which we can analyse the problem at a greater depth. Herein the problem is detailed according to its various aspects. The two essential aspects of the problem are exceptions and multiple inheritance. These two aspects have formed the spearhead of research. Having described these two aspects, they are then described as they occur in particular systems. Although there have been many knowledge-based systems which have, consciously or unconsciously, employed inheritance, three stand out as being of particular importance. These three are Fahlman's NETL [Fah79], NUDGE and FRL by Roberts and Goldstein [GoR77, RoG77] and Bobrow and Winograd's KRL [BoW77]. The final section in this chapter is an introduction to Touretzky's work, which provides a mathematical theory of inheritance.

### 2.2. A statement of the problem

#### Exceptions

In an imperfect world, there are many things which exist as exceptions to classifications or concepts. Without the ability to cancel properties in general, a

semantic network cannot represent exceptions, which is a serious drawback just because they are such a common aspect of our world. We must find some way of representing them in inheritance systems. The earliest systems simply enforced inheritance on all nodes below the one in which the property was stored; there were no exceptions because inheritance of properties was a necessary property of the tree-structure around which they were built. Clearly this approach is inadequate for anything other than very simple knowledge domains.

Suppose we tried to build a *rigid* taxonomy for mammals, stating that nodes below the current one must inherit its properties. If we start right at the top with the *mammal* node, what properties could we place there being confident that all mammals would inherit them? One possibility is "vertebrate." "Warm-blooded" and "viviparous" seem likely too, but there are exceptions to each. The bat is both warm and cold-blooded, and the duck-billed platypus is not viviparous.

Instead of this hopeless scheme, most inheritance systems allow particular individuals to override the inherited properties which do not describe them. So, any individual can accept as many of the inherited properties as are pertinent, and override the others. Classes, such as *bird*, may make an exception to properties too. The way to allow exceptions to property inheritance is to introduce specific cancellation. As soon as we do allow exceptions, the logical properties of the hierarchy change, and it has taken a long time to make these changes explicit [Bra77, Tou86]. Reasoning becomes nonmonotonic because new knowledge can

make hitherto valid facts invalid, thus simple hierarchical representation is no longer appropriate.

Consider the simple taxonomic inheritance system in Figure 2.1 in which an individual, "Ziggy," is shown to be a member of the "mole" class. Moles have the property of being black. Ziggy however, (due to an accident with a paint-pot,) is orange. In the simplest of inheritance reasoners, when we ask the system to find Ziggy's colour, it will find orange and stop, proceeding no higher up in the IS-A hierarchy thereby failing to find the default information that moles are usually black, and in turn failing to find the contradiction; namely, that Ziggy is both black and orange.

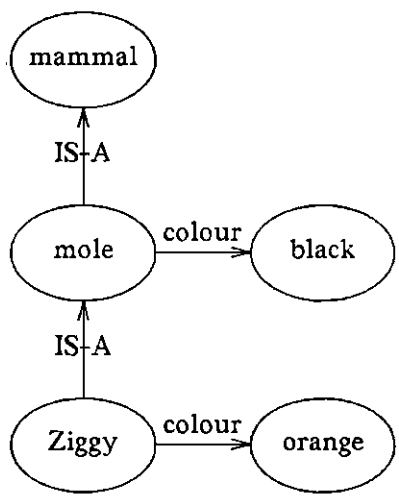


**Figure 2.1 A simple hierarchy containing a contradiction**

The solution seems to lie in permitting direct negation of the "blackness" property of moles, but Touretzky notes,

> Partly due to the idiosyncratic way in which exceptions are handled in present inheritance systems, these systems [FRL and NETL] have avoided any explicit representation of negation. There is no way in FRL, for example, to say that mammals are not feathered. Such negative assertions should be inheritable, just as positive assertions are. If we say that mammals are not feathered, we should be able to infer that humans are not feathered, elephants are not feathered, Clyde is not feathered, and so on [Tou85].

A class such as *penguin* should be able to specifically cancel the default property of the superclass *bird* that a bird can fly. This is important because it prevents incorrect inheritance of values, although we may have no alternative value to replace the one we have blocked. For example, bananas are usually yellow, but when we find an exception, we may not know its colour, though we do know the colour is not yellow.

This system of blocking the information from higher up in the hierarchy with more local information has its problems. The blocking values used instead of the inherited values add confusion when the system tries to access the network. It is not clear from where the new value which blocks the inherited one came if *it* was not inherited. Also, exceptions of this nature will only work for properties and not for cases where inherited membership of a class needs to be cancelled. There is no way to say that an individual is not a member of a class once that individual appears below that class in the hierarchy. Class membership cannot be blocked: there is no 'value' which will block membership. But all these problems can be reduced if we use explicit cancellation [FTR81]. NETL [Fah79] is

one of the few systems that deals with the problem of exceptions. Despite Fahlman adding a specific link to cancel a property to NETL, there were always cases for which cancellation would not work correctly. He realised that there was something wrong fundamentally, and eventually had to rewrite the system using both *IS-A* and *IS-NOT-A* links.

It is worth pointing out that there is a distinction between assertions which are necessarily true, and those which are merely assumed true by default. Necessary truths are always true and could never be false; necessary truths are true in all possible worlds. Defaults are only contingent truths however, and it is logically sound to negate them. This distinction is important because knowledge representation systems must recognise it in order for inheritance to work properly. For example, a necessary truth is, "All men are mortal." In a system which can deal with negative statements, there should be a mechanism to stop a particular human from cancelling this necessary truth, and thereby becoming an immortal human. The system must be able to distinguish between necessary and contingent truths. Contingent truths, such as "All aeroplanes fly" may be excepted. In a network, therefore, there will be some IS-A links which are not defaults at all, and should not be cancelled because they could never be false. Most often, IS-A is taken to be default, and therefore represents a property which could be cancelled.

### Multiple inheritance

When inheritance systems had developed beyond the simplest hierarchies, they allowed a class or an individual to inherit from multiple superclasses. This was a distinct advantage over the tree hierarchies, but new semantic problems emerged. For the first time the network might contain two theories which, although independently valid, created a contradiction. Quite what an inheritance reasoner was to do about such a problem remained a mystery, and the problem was not solved until Touretzky presented the inferential distance ordering [Tou86]. This scheme is able to order the proofs about classes and individuals, rather than trying to order the objects themselves as simple IS-A traversal schemes do, and thereby dealing with multiple inheritance. The solution to the multiple inheritance problem comes in finding an ordering of the (conflicting) valid proofs such that one may be chosen over another. Below are several examples which show the deficiency of earlier inheritance systems faced with this problem which tried to order proofs according to the length of the inference path to each proof. Touretzky's solution is to provide a new way of measuring inferential distance. He says that, "the essence of this ordering is that an individual or class A is 'nearer' to class B than to class C iff A has an inference path *through* B to C [Tou86]." In Touretzky's words, inferential distance is "a measure of 'between-ness'."

Figure 2.2 below shows how multiple inheritance can lead to multiple correct proof sequences. Using Touretzky's notation [Tou86] the ordinary arrows

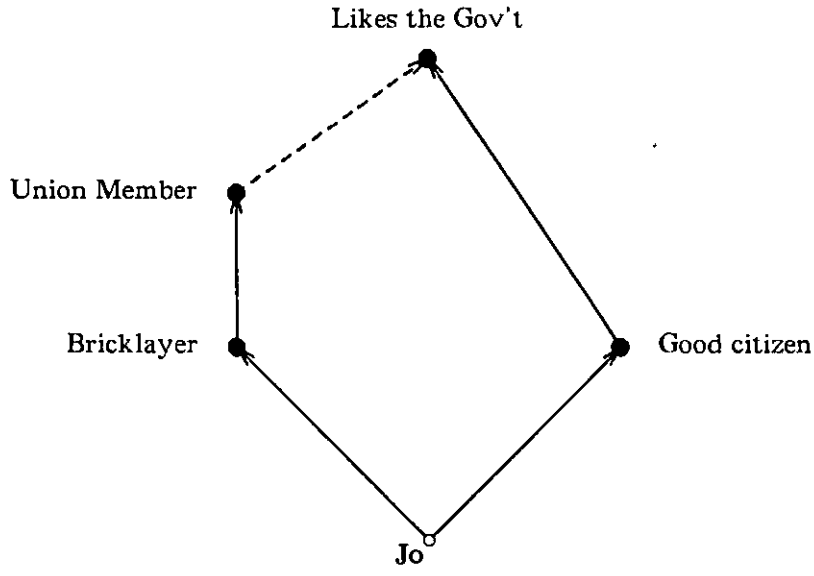


**Figure 2.2 A difficulty with multiple inheritance**

are IS-A links whereas the dashed arrows are IS-NOT-A links.[1] Solid circles are classes and open circles are individuals. Two proof sequences can be generated, one showing that, "Jo likes the Government" and the other showing that she does not. With Touretzky's inferential distance ordering, neither proof is invalid because 'good citizen' is neither a subclass of 'bricklayer' nor of 'union man'; the inferential distance ordering algorithm only allows one class to override another if the two are in a strict superclass/subclass relationship. Both proofs are valid in themselves, but are mutually inconsistent. A simple reasoner like FRL would conclude that Jo both likes and dislikes the government at the same time, which

[1] In Touretzky's notation the IS-NOT-A links are cross-hatched.

is inconsistent. NETL's algorithm chooses between the two opposing proofs on the basis of which inference path is shorter, which is irrelevant to which proof is the correct one. The important thing about Touretzky's algorithm is that it does not prefer either proof over the other because there is insufficient information to make a decision. Consider another problem case.

*Dogs are torpid.*

*Useful dogs are dogs.*

*Useful dogs are not torpid*

*Guide dogs are useful dogs.*

*Pete is a guide dog.*

These statements are represented in Figure 2.3 below. The reader is asked to bear in mind that this hierarchy is extremely unnatural, for classification according to torpidity is by no means normal. But for the purpose of illustrating the difficulties of multiple inheritance, the example is illuminating. According to the inferential distance ordering, Pete is nearer to guide dog than to dog since he has an inference path through guide dog to dog but not the other way round. An inheritance reasoner which employed the inferential distance ordering would conclude that Pete was not torpid. Two statements about Pete can be argued from this diagram. It can argued that Pete is a guide dog, guide dogs are useful dogs, useful dogs are not torpid, therefore Pete is not torpid. But we also have the proof that Pete is a guide dog, guide dogs are useful dogs, useful dogs are dogs, and dogs are torpid, therefore Pete is torpid. The conclusion should be that Pete

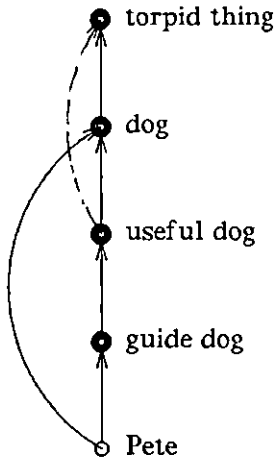


**Figure 2.3 Another problematic inheritance hierarchy**

is not torpid. Both FRL and NETL draw the correct conclusion by preferring exceptions at a lower level than inheriting from a superclass. But if we add the (redundant) statement that "Pete is a dog", we can then prove that since dogs are torpid, Pete is torpid. This statement ought to have no effect on the conclusion that an inheritance reasoner reaches. However, FRL and NETL fail to handle this network correctly. Pete inherits from two superclasses: dog and guide dog. So, in FRL, we would find that Pete is both torpid and not torpid, while in NETL, the shortest inference path is taken, and so we would find that Pete is torpid, using the path provided by the introduction of the redundant statement.

It should be clear from this that we cannot use path length alone to choose between competing inferences. Such a scheme works where inheritance is from

one superior only, but is inadequate for multiple inheritance. Touretzky [Tou86] suggests that one way to repair these inheritance reasoners is to ban redundant statements from being asserted. So for example, we would ban the statements "Pete is a dog" and "Pete is a useful dog." He goes on to point out that it would be most unsatisfactory for any knowledge representation scheme to ban statements which are consistent and true, especially when they could be derived. However, the system might allow any statement to be entered, and then check its knowledge base to see if it could be derived before asserting it. If the new statement is derivable, then there would be no need to assert it. Thus, the system would accept "Pete is a dog," but not change the knowledge base at all since this statement can be derived from existing facts. If any statement is consistent with the knowledge base and not redundant, (i.e. could not be derived,) then it would be asserted.

Touretzky points out another reason for not banning redundant statements:

> Finally we cannot easily ban redundant statements because there *are* no truly redundant statements in a system that allows exceptions. A statement that supplies redundant information about a class as a whole may yet have an important effect on subclasses [Tou86].

Here is an example of what Touretzky has in mind:

*Cheeses are tasty*

*Tasty things are good to eat*

*cheeses are good to eat*

*Stale cheeses are cheeses*

*Stale cheeses are not good to eat*

These statements are depicted in Figure 2.4. The third statement, that cheeses are good to eat, seems to be redundant because it can be deduced from the first two. However, it may be that we have another reason for believing cheeses are good to eat, such as the result of an inquiry into dairy produce. In such a case the statement does have import beyond the current set of statements. But are these redundant statements *necessary* to the system? The system would not know that statements had extra significance if they were asserted directly rather than being derived; there should be no difference in significance between a statement that is derivable and one that is directly asserted. Thus even if we admit that redundant statements might have greater import than if they were derived, the system itself would not be aware of it. However, Touretzky's inferential distance order-

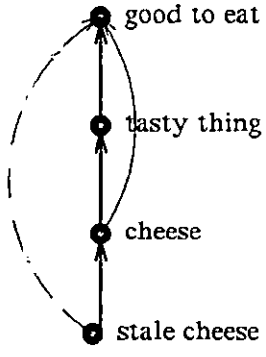


**Figure 2.4 A hierarchy with a redundant statement**

ing is not disturbed by redundant statements because it does not consider the length of the inference path.

Multiple inheritance gives rise to a second problem otherwise not manifest in networks: ambiguousness. Many systems that permit multiple inheritance are unable to tell if their networks are ambiguous. For example,

*Those whose hearts have stopped are dead*

*Those whose brains are functioning are not dead*

*Harold's heart has stopped*

*Harold's brain is still functioning*

How may we decide from these axioms whether or not Harold is dead? As has been shown, FRL would conclude that Harold was both dead and not dead, which is contradictory. The conclusion of NETL would depend upon the idiosyncrasy of its marker propagation algorithm. The correct conclusion is surely that a decision cannot be made on the basis of these facts alone; they are truly ambiguous. One theory should not be preferred above another.

## 2.3. Previous solutions and their shortcomings

Below is an overview of systems which have tried to deal with inheritance. They have been selected from a divers range because they are considered the most important. The systems are described and their deficiencies are listed.

### 2.3.1. Fahlman's NETL

#### Introduction to NETL

Fahlman calls NETL a "system for representing and using real-world knowledge." It consists of two parts. First, there is a parallel network memory scheme. Knowledge is represented by a network of simple parallel processing nodes which store about twelve marker bits. There are links which can propagate these markers from node to node, in parallel throughout the network. Knowledge is retrieved from the network by moving the markers, which constitutes some form of search. Fahlman claims that the search time and the time for common deductions is both fast and constant, irrespective of the size of the knowledge base. The second part of the system is a vocabulary of conventions and processing algorithms—a sort of language—for representing knowledge as links and nodes. This language is given the name NETL—an acronym formed from the two words NETwork Language.

Fahlman claims advantages of NETL over earlier systems such as KRL and NUDGE:

(a) Real world concepts are represented more efficiently and precisely than in earlier systems due to consideration of representational techniques.

(b) There is a "single, clear, explicit organising concept." Any knowledge-base system must not only store and retrieve single facts, but must also allow the user to make copies of the descriptions in memory. The knowledge system

behaves as if a chunk of the semantic network has been copied, but in fact there are no redundant copies in memory. Everything related to a description, such as its links, its sub-parts and so on, are inherited in their entirety by the copy. Fahlman makes the claim that NETL is unique in stating this goal explicitly.

(c) NETL is designed to operate on a parallel network machine. However most of the ideas in the system are applicable to knowledge based systems in serial machines as well.

**The overall goals of NETL**

Neither of the two parts of NETL is dispensible. The language NETL is the part which contains the conventions and procedures to exploit them. Without these, the parallel network system is not useful. The procedures fall into three categories. The first category deals with incoming knowledge, and the translation of it into bits of network structure. During this process, checks are made for consistency and compatibility. The second set of procedures accesses the existing network in response to requests for information and matches. It basically performs all the searching and matching by means of the marker bits at the various nodes. The final set of procedures does all the book-keeping for the system.

Behind the scenes of NETL are all the representational conventions. Fahlman says that the real substance of NETL is its "integrated collection of viewpoints and attitudes." Unfortunately however, we are nowhere treated to an

exposition of precisely what these viewpoints and attitudes are. One of the design aims of NETL was to improve upon its predecessors such as KRL. Whilst KRL is a language for creating knowledge-based systems, NETL is merely a language for representing knowledge. This indicates a fundamental difference in design philosophy, for in KRL, the user is able to make many choices regarding how a certain chunk of information should be represented and how it should be retrieved. NETL allows the user no such freedom. *Prima facie*, it might appear that KRL has the advantage. However, this is not the case because NETL was designed so as to make such decisions by the user unnecessary. It is rather like the difference between a pre-programmed calculator and a programmable calculator. The first comes equipped with all the programs for calculating integrals, factorials and so on, whereas the latter provides one with the programs to enter oneself as and when they are needed. NETL is like the pre-programmed calculator. The main advantage is that the user does not have to know so much about the system, and remains much more of an external *user* rather than an expert programmer. Fahlman's reason for depriving the user of making representational decisions is that the parallel network system is so powerful that tinkering or hand-tailoring of the processing strategies is redundant.

The five basic aims of NETL which have been stated implicitly above are:

(1) *Compatibility with the parallel network implementation.*

NETL was the first inheritance system to be designed from scratch with parallel mechanisms in mind.

(2) *Completeness: the system should be able to represent anything that people can.*

Fahlman hoped that NETL would be complete, that there would be no domain in which it was representationally inadequate. At the time the book was written [Fah79], Fahlman claimed not to have found any area in which NETL was fundamentally unsuitable. Of course there were areas in which it was inelegant and ungainly, but the system could still represent the knowledge, albeit in this fashion. However, Fahlman says, "as new domains are explored, it is likely that problems will be encountered which will invalidate some of NETL's basic ideas, but I see no signs that this is imminent."

(3) *Semantic precision*

As pointed out by Woods [Woo75], many early semantic networks suffered from various forms of semantic imprecision. For example, the precise function of a link has often been left inexplicit and indefinite. Fahlman believes that NETL avoids most of the problems noted by Woods. The main advantage NETL has over other systems which have been tarred by Woods' brush is that it has the concept of an explicit virtual copy which makes the semantics entirely open and clear.

(4) *Simplicity and intuitive clarity*

In order for knowledge to make its way into the system, the representations used by the system should be both natural and intuitive. The design and representational philosophy behind the system should also be pellucid:

everything must be open and laid bare before the eyes of the user. If a certain piece of information can be represented in several ways, it should not matter which representation is chosen in terms of efficiency and clarity. The user should be oblivious to the possibility of speeding the system up by choosing between representations. All the alternative representations should lead to comparable results. These idealistic goals seem to be too grand to be realised however, and NETL is an unhappy compromise.

(5) *Economical representation*

It will be gathered from the previous design aims and discussion, that efficiency was never one of Fahlman's driving concerns. In building NETL, issues of clarity and representational adequacy were always placed before issues of speed and frugality of link and node use. One reason why he could do this was that NETL was well ahead of any competition because it was built upon a parallel hardware system. It turned out to be a surprisingly fast system. So while the goals of minimising link use, and minimising inference times were on Fahlman's mind, he placed the other design goals before these two.

**NETL and the inheritance problem**

Fahlman was the first person to expressly design hardware, and parallel hardware at that, for implementing a network. Marker propagation algorithms and detection mechanisms eliminate much of the costly search intrinsic to previ-

ous von Neumann systems. The *virtual copy* is what makes inheritance function in Fahlman's system. This dominant concept orders the information represented. It is a convenient way to construe inheritance in semantic networks because it allows us to assume that all the properties at a parent node are accessible to its subnodes.

Humans can easily endow individuals with a lot of knowledge inherited from a general class of objects. Thus if one were told that Sarah has broken her leg, one immediately has an image of her with a plaster-cast and hobbling about on crutches. One would be able to answer questions such as "Is she able to play baseball next Friday?", and "Why does she always wear skirts now where she wore trousers before?" All of this reasoning is performed with exasperating effortlessness. What is surprising about our manipulation of concepts is that we are able to extract particular information from them. For example, we are able to discern very quickly whether we know a particular fact or not. If one were asked if one knew how fast a cheetah can run, one can instantly discern whether or not one knows that fact. How do we manage to produce an answer so quickly when we know such a large number of facts? Clearly, we cannot search through our entire stock of facts and draw a negative as to the answer to the question; what we do not do is try to match the question to answers in our minds. Such a procedure would be intolerably long—even for the brain's massively parallel mechanisms.

It is relatively easy to represent explicit knowledge in computers, and there are many systems and languages which do this. The only way to access explicit knowledge however is by searching and matching. For example, we might have a large number of facts stating explicitly what John drinks. We could find out what John drinks by searching the knowledge-base for these assertions and retrieving them. Whilst ideal for explicit knowledge, these same systems are not adequate for representing and retrieving implicit knowledge. If we know facts about some generic "cow" and we are trying to find out about a particular cow Ermintrude, we have to have some deductive mechanism for relating the two. To find out everything about Ermintrude that is not explicitly stated, we must search the semantic network for knowledge about her that can be deduced from other knowledge. A very large number of related concepts have to be searched to make sure that we have found all the knowledge relating to Ermintrude. NETL sets out to solve the problem of how this search can be performed in a reasonable amount of time.

Fahlman resigns himself to performing an extensive search. In order to do it quickly, a lot of parallel computing power is needed. The parallelism is achieved by storing the knowledge in a semantic network which is constructed from link units and node units alone. The node units represent concepts and entities in the knowledge base, and link units represent relationships between nodes. The deductive search is performed by virtue of the marker bits which are propagated by the two types of unit. However, their propagation is strictly controlled by an

external computer ycleped the network controller.

Concepts are represented by two types of nodes: (a) individual nodes[2] and (b) type nodes. The type nodes are meant to be "template descriptions" from which any number of individual copies may be made. A type node serves as the epitome of a particular set of individuals. Thus the type node for a football game includes anything we might say about the "typical football game." Every individual node which is a football game will inherit all the properties of the paradigmatic game unless they are explicitly cancelled. The problem with this scheme is that we still do not know what the type node ought to represent. If it is trying to represent a concept, as Fahlman wants it to, then we run into difficulties trying to specify the necessary and sufficient conditions for that concept. That NETL requires all the nodes in its network to be defined and carefully honed to fit is a severe weakness of the system. Ideally, we would like a system to be able to represent knowledge about which it had no previous concepts whatsoever. So, if one wanted the system to represent the concept of a man with bananas growing out of his head, one expects the system to be able to deal with this. But NETL would have great difficulty doing this even if it already knew about bananas and heads.

These worries did not stop the inheritance system in NETL from functioning reasonably well and Fahlman gives the following example. Suppose Clyde is an elephant. He inherits properties from the typical-elephant node through the

[2] These are token nodes in Quillian's notation [Qui66].

virtual-copy relationship. Once the virtual copy has been created, a "type-of" link is established between elephant and mammal. The "typical-elephant" node will inherit all the properties of the typical mammal. For example, if typical mammals are viviparous, then so are typical elephants. If typical mammals are warm-blooded then so are typical elephants. Elephants are thus a subset of the class mammal. A property which exists at some node $n$ is meant to be inherited by all descendants of $n$, unless explicitly cancelled.

Suppose Taylor's market a certain port, and someone asks what colour it is. The system might have a colour link from "typical-port" to ruby. The parallel hardware might have to search back up the hierarchy quite some way before finding the "typical-port" node and its associated colour. What the search does is mark every node above the individual-node "Taylor's port", which might have properties which Taylor's port should inherit. What happens next is that a general command is issued to all "colour" links that are attached to any of the marked nodes. If nothing goes wrong, the only "colour" node which reports is *ruby*. Although we do not find out where in the tree this colour node is, we do not need to; the required colour has been inherited from somewhere higher up in the hierarchy.

One of the difficulties of using type nodes is that the system designer must know about the type beforehand in order to define it. As has been said, the system should be able to accept concepts and assertions about which it has no existing knowledge. This problem cannot be overcome if concepts are used at the

representational level.

**Exclusive splits and clashes**

NETL will allow us to represent knowledge, and retrieve it. But how does it handle inconsistencies? If the system contains some knowledge about the world already, how is it to deal with glaring inconsistencies which come in the form of requests to store new information in the semantic network? The kind of inconsistencies which a knowledge representation system handles are obvious and egregious. For example, if the system knows that hemlock is poisonous, and a new fact comes in that hemlock is a good ingredient for casserôles, the system should be able to detect these inconsistencies. The system should also recognise that there is a problem with phrases such as, "The trees are happy", or "The stones were shouting obscenities", or "Julia is proud of her height." Fahlman believes that most clashes of this type arise from the structure of the knowledge base and the operation of the inheritance mechanism.

One possible solution to the inconsistency problem is to state explicitly that hemlock is not a good ingredient for casserôles, that rocks do not shout and so on. But this is not the best solution because we would have to write so many of these "X is not Y"-type facts. What we need to do is separate plants into edible and inedible, and indicate that these two sub-classes are mutually exclusive. If an attempt is made to create a set containing members from both sets, such a violation could be detected, and a complaint made to the process asking for its crea-

tion.

### 2.3.2. FRL and NUDGE

#### Introduction

NUDGE and NETL are reviewed in full in [GoR77, GoR79, RoG77]. NUDGE was the system used by Goldstein and Roberts to try out their first frame representation language:FRL-0. Based on this research, they wrote FRL, an improved frame representation language.

There are two approaches to knowledge-based reasoning systems. First, there are systems which rely on a large store of specific knowledge to reason. Second, there are systems which use a small set of general reasoning mechanisms. NUDGE makes use of this latter knowledge-approach rather than the power approach. It has a large database of knowledge in order to expand informal scheduling requests. The knowledge base is provided in order to "fill in the blanks" of some request. It provides the missing details, resolves inconsistencies, determines options, identifies prerequisites and plans for the unexpected. The language FRL-0 was written to manipulate the knowledge base. Traditionally, properties have been described only by attribute/value pairs. But in FRL-0, they can also be described with defaults, comments, abstractions, constraints, indirect pointers from other properties, and attached procedures.

Using attribute/value pairs, facts about some object could be conveniently held on a LISP-like property list. The attribute name is then a pointer or a link

to its value. For example, if we were thinking about an engine specification, we might represent some of its properties as follows:

```
Engine X
        cylinders   8
        cooling     Air
        stroke      4
        fuel-type   petroleum
        capacity    1543
```

So the attribute 'capacity' has the corresponding value of '1543'. Whilst attribute/value pairs are adequate for simple declarative sentences, anything more complex creates major representational difficulties. For example, how might the sentence, "Engine Y has more than four cylinders" be represented? If we represent such a sentence as:

```
Engine Y
        cylinders   (greater-than 4)
```

we are no longer making simple declarations. The attribute remains the same and may be seen as a pointer to a value, but the value has now become the predicate "greater than four". This is a problem because the attribute has now become a complex predicate, and is not consistent with the attribute/value representation. One way out of this difficulty is to maintain attribute/predicate pairs rather than attribute/value pairs. So, in the first example, the values "four," "eight," "air," "petrol" would become the predicates, "is four," "is eight," "is air," "is petrol." Woods [Woo75] declares that "there are at least two possible interpretations of the meaning of the thing at the end of the link— either as the name of the value

or as a predicate which must be true of the value." The first of these two possible interpretations is hardly an alternative since it cannot be used to represent predicates such as "greater than four cylinders."

Suppose we needed to represent a still more complex sentence: "Engine X has a larger capacity than Engine Y." This poses further problems. The link *capacity* can still be used as a pointer within the property list of Engine X, but it now points to a complex predicate "has larger capacity than Engine Y." It seems that there is no way to avoid a reference to Engine Y within the property list of Engine X. Reference to other objects as the value of some attribute is cumbersome and awkward. We do not want to have to modify our data structure according to each new item we try to represent. If we want to abandon the idea of attribute/value links, we might decide to write:

(larger (capacity (Engine X)) (capacity (Engine Y)))

An alternative proposed by Woods is to say that the *capacity* link points from Engine X to a node which represents the intensional object "the capacity of Engine X." Similarly, we would have a capacity link from Engine Y to an intensional node representing "the capacity of Engine Y." A relation *larger* could then be established between these two intensional nodes.

In this case we would have to make major changes to the semantics of the networks, which was precisely Woods' point. At the end of each attributive link there must be an intensional node, and predicates must be asserted which are true

of the objects at the intensional node. Also, a record must be kept to indicate that this new node was created to represent the concept of the capacity of Engine X. Being greater than the capacity of Engine Y is not a defining property, but a separate assertion about the node. Thus Woods has made a distinction between defining and asserting properties of a node.

Let us now return to the discussion of NUDGE. In order to build the reasoning system, a problem domain had to be chosen in which informal requests were made and which was small enough to be representable in a knowledge base. The most suitable domain for the experiment was an ordinary office, in particular, the scheduling needs of a manager in that office. The manager has to coordinate a team, each member with his own responsibilities, and most important, he has to manage his own time, arranging for meetings, going to conferences and so forth. Could an intelligent system help with these difficulties? Goldstein and Roberts thought that it could, and built NUDGE. The principle idea was not to build a system to aid a manager, but to see if a system could reason intelligently about a limited problem domain. NUDGE was required to schedule meetings, resolve conflicts, issue progress reports, and alert the manager to deadlines.

The activities of the office are represented in NUDGE by hierarchies which involve information transfer. There are hierarchies for the activities themselves, for those involved in the information's transfer, for the plans which govern these transfers, and for the additional demands on time, space and personnel. Each hierarchy contains about five levels and has about one hundred objects, each

described by a generalised list of properties, which we now know as a frame, thanks to Minsky [Min75]. Minsky's motivations for proposing frames was his belief that intelligence arises from large amounts of highly specific knowledge rather than a few inference mechanisms applied to a variety of domains. Goldstein and Roberts' motivation for using frames was that an intelligent system can be achieved through the use of a library of frames where each frame is a packet of knowledge that provides a description of a typical object or event. These descriptions contain both an abstract template providing a skeleton for describing any instance and a set of defaults to describe typical members of the class. The defaults enable the information system to supply missing detail, maintain expectations and notice anomalies.

**NUDGE in use**

A typical request made by a manager to NUDGE might be:

*S1: Schedule a meeting with Bruce for next Tuesday.*

S1 is a very normal request which NUDGE is expected to recognise and expand into a formal request, all the unspecified points being specified by NUDGE:

*S1': Schedule a meeting between Bruce and me at three in the afternoon next Tuesday, one week from tomorrow, lasting one hour in my office to discuss the personal assistant project.*

The formal request causes NUDGE to build a collection of instantiated frames containing all the information about the request. This collection is called a *frame*

*gestalt*. The whole request ends when NUDGE has converted the frame gestalt into a calendar showing potential clashes and a set of procedures for resolving the conflicts. For example, if the meeting cannot be scheduled as requested, then NUDGE is flexible enough to consider making appropriate changes to it. It may be that the manager making the request already has a meeting scheduled at the time he wants to meet Bruce. NUDGE is able to substitute the manager for a member of his team to resolve conflicts where such a substitution is appropriate. NUDGE has been able to make these changes by knowing that the manager is not essential for the meeting with Bruce, and that any of the members of his team could see him. This flexibility arises from its access to a wide variety of scheduling strategies. Only as a last resort will NUDGE use a sledge-hammer to crack a nut and start shuffling the calendar to fit all the meetings in.

**The data structure: frame gestalt**

A gestalt structure is so integrated as to constitute a functional unit with properties not derivable from its parts in summation. A frame gestalt is a structure built up from a set of generic frames, with particular values set appropriately for a specific request. In the initial request there are 'clues' which NUDGE uses to select the generic frames. The first task of NUDGE is to choose the right frame gestalt for the request. Whatever information is missing from the request is computed from defaults, constraints and procedures which are associated with the frames. Having chosen the right frame gestalt, NUDGE has to compute the

values of slots which have not been filled in by the original, expanded form of the request. To put it another way, NUDGE has to complete a frame gestalt in its entirety where the input has been a set of frames partially completed by information extracted from the original request.

The process of expanding the frames extracted from the initial request is aided by using techniques available through FRL-0: comments, abstractions, defaults, constraints, indirection and procedural attachment. Comments provide guidelines for the reasoning system, enabling it to judge the reliability and strength of various constraints and to remove inconsistencies which arise from conflicting contributions by different frames during the construction of the frame gestalt. An important aspect for this research is how the system would allow defaults to be overwritten. Comments provide the answer; they tell the controlling program to overwrite the default suggested by generic knowledge with more parochial knowledge. Comments are also useful if a user of the system wants an explanation of how a particular conclusion has been reached. Annotation enables the system to explain inferences.

**The inheritance mechanisms of NUDGE**

Accompanying the hierarchy of frames is the concept of maintaining default information. The slots of some general activity typically supply default information for questions or requests which occur frequently. The use of defaults provides NUDGE with its power and flexibility. In forming a frame gestalt,

defaults from a superior frame are used unless they are overridden by information from a more reliable source. A knowledge representation language such as FRL-0 must describe properties if it is to support recognition of new instances of a generic concept. Constraints can be attached to any slot within a frame to specify requirements and preferences. A requirement is a fixed condition in the general frame which must be met by a particular instance, but preferences are flexible rather than rigid. Defaults are really preferences which recommend specific alternatives rather than being predicates which act on some set.

FRL-0 supports two types of inheritance. Additive inheritance allows new facts to be added to an incomplete concept by one higher up. Restrictive inheritance allows specific information about an individual or class to override the more general concept. Goldstein and Roberts used comments which the inheritance mechanism read to see how far up in the hierarchy it should go.

One important point about defaults and inheritance is that a slot may receive conflicting values from the inheritance mechanism and from the default frame. Which of these two values should be used? Goldstein and Roberts avoid the subtleties of this problem by simply forcing the system to take inherited values before default values. Thus, before choosing a default value, the whole hierarchy is scanned for a suitable value to inherit. Realising the inadequacy of this method, Goldstein and Roberts compromised by providing an inheritance override, such that the user may obtain full control of the inheritance mechanism. The user can request the "heritage" of the slot; that is, all the values filled in the

slot and its superiors and from whence they came. On the basis of this information, the user may force inheritance of the value which is most appropriate. What we really want is for the machine to be able to discriminate between the possibilities and choose the most appropriate one without user intervention. Ideally the user should never see, nor have reason to doubt, such a mechanism.

**Lessons from NUDGE and FRL**

FRL-0 has considerable power through the addition of comments, constraints, defaults, procedural attachment and inheritance, all of which make it much more attractive than the simple attribute/value pair representation schemes. FRL-0 was an experiment in the utility of frames. NUDGE is one step towards an artificial intelligence system with common sense reasoning. By generating frame gestalts, the system minimises the possibility of overlooking obvious alternatives. Using defaults and preferences means that a lot of the work is done implicitly. Some tolerance of minor inconsistencies is a benchmark of a robust knowledge system. One of the main results of NUDGE was the discovery that hierarchies of concepts are very useful because fine distinctions can be made and because it aids the recognition of new objects. The shortfall of NUDGE is that it uses concepts in a liberal sense. It is the author's opinion that such use of concepts is a major contribution to the problem of inheritance.

### 2.3.3. Bobrow and Winograd's KRL

The purpose of KRL was to define a knowledge representation language for use in understanding systems [BoW77]. It was an attempt to integrate procedural knowledge with a broad base of declarative forms. These forms provide several ways of expressing the logical structure of knowledge. This in turn lends flexibility in associating procedures for memory and reasoning with specific pieces of knowledge, and allows control over the relative accessibility of different facts and descriptions. Declarative knowledge was represented as *objects* with associated *descriptions*. The objects form a network of *units* whose links are variously sorted, each having certain implications for the retrieval process. A procedure may be directly associated with the internal structure of a conceptual object. Such procedural attachments mean that the characteristics of the specific entities involved determines the operation steps.

Bobrow and Winograd laid down some fundamental intuitions:

(1) Knowledge should be organised around *conceptual entities* with associated descriptions and procedures.

(2) A description must be able to represent partial knowledge about an entity and accommodate multiple descriptors which can describe the associated entity from different viewpoints.

(3) An important method of description is comparison with a known entity with further specification of the described instance with respect to the prototype.

(4) A stored set of expected prototypes dominates the reasoning mechanism. New objects and events are compared to the prototypes, and specialised reasoning strategies connected to these prototypes are initiated.

(5) Information should be clustered to reflect use in processes whose results are affected by resource limitation and differences in information accessibility.

The natural organisation for declarative knowledge is to centre it around a set of *conceptual entities* with associated *descriptions*. Three underlying operations in the system are *augmenting* the description to incorporate new knowledge, *matching* two given descriptions to determine if they are compatible for the current purposes and *seeking* referents for entities that match a specified description. A description is made up of several descriptors, each corresponding to a different viewpoint. Entities may be described by comparing them to other entity descriptions already in memory. The object used as a basis for comparison is called the *prototype* and it provides a *perspective* from which to view the object being described. The details of the comparison can be thought of as a *further specification* of the prototype. In describing an object by comparison, the standard for reference is often not a specific individual, but a stereotypical individual which represents the *typical* member of a class. Such a prototype has a description which may be true of no individual in the class, but combines the *default* knowledge applied to members of the class in the absence of specific information. A single object or event can be described with respect to several prototypes with further specifications from the perspectives of each. The further specifications in

a description by comparison can provide more detail to go along with less specific properties associated with the prototype, or can contradict the default assumptions which are assumed true in the absence of more specific information. For example, the destination of a trip might be the default 'city', and in a particular event is given as Bangkok. The default for a trip might also include the fact that the traveller starts from place A and returns to place A at the end of the trip.

Of course, specific instances might violate these defaults. Any specific instance automatically assumes the properties of the prototype unless they are explicitly denied. There is considerable evidence to show the importance of making use of typical and expected properties in contexts in which the reasoner has incomplete information about the world and cannot prove logically that a particular individual has a desired property [Rei81, Rei83, Tou84]. It is important to see this analysis as an intuition about how people structure descriptions rather than as a specific technical device. Many of the mechanisms proposed for knowledge representation, such as semantic networks and frames, can be used in a style compatible with this kind of inheritance of properties.

KRL associates procedures with a class, represented by a unit. This means that they can be linked to the slots of a unit and specifically associated with several different descriptor types. This makes the clustering of the procedures correspond better to the conceptual structuring of the domain. In addition, KRL provides through its use of perspectives, a notion of subclass that allows objects to inherit procedural as well as declarative properties. Since each unit can contain

multiple perspectives, it can be a member of a number of subclasses.

### 2.3.4. Touretzky's solution

When considering a path of inheritance, most reasoners apply the simple rule that the inference path required is the shortest one. The inadequacy of this approach was apparent when we considered multiple inheritance, redundancy and ambiguity in previous sections. Touretzky [Tou86] claims to have found a partial ordering of the default properties held in an inheritance network which correctly performs inheritance even in the problem cases of inheriting from multiple superiors, redundancy and ambiguity. This ordering only allows one class to inherit from another if they are in the correct subclass/superclass relationship. Most simply, this ordering says that A is a subclass of B if and only if there is an inheritance path between them. The concept which is behind the inferential distance ordering is that an individual or class A is 'closer' to B than to C if and only if A has an inferential path through B to C. Touretzky says that inferential distance "is not a measure of length; it is a measure of 'between-ness'." With this measure, redundant statements are not merely semantically redundant, they are also redundant for the inheritance reasoner; it can simply ignore them. Since this is the case, if an inheritance system uses inferential distance ordering, there is no need to exclude redundant statements from the database.

It is possible that a system which uses the inferential distance ordering may have no criteria for deciding between competing inference. The correct action in

such a case is surely to make no choice, but nevertheless to evaluate the paths available. Consider the following case, which is similar to the ambiguity discussed above.

*medicinal drugs are not bad for you*

*addictive drugs are bad for you*

*tranquilizers are medicinal drugs*

*valium is a tranquilizer*

*valium is an addictive drug*

Any theory of inheritance that is to work must be able to detect this ambiguity and take sensible action. The representation of these facts within a knowledge

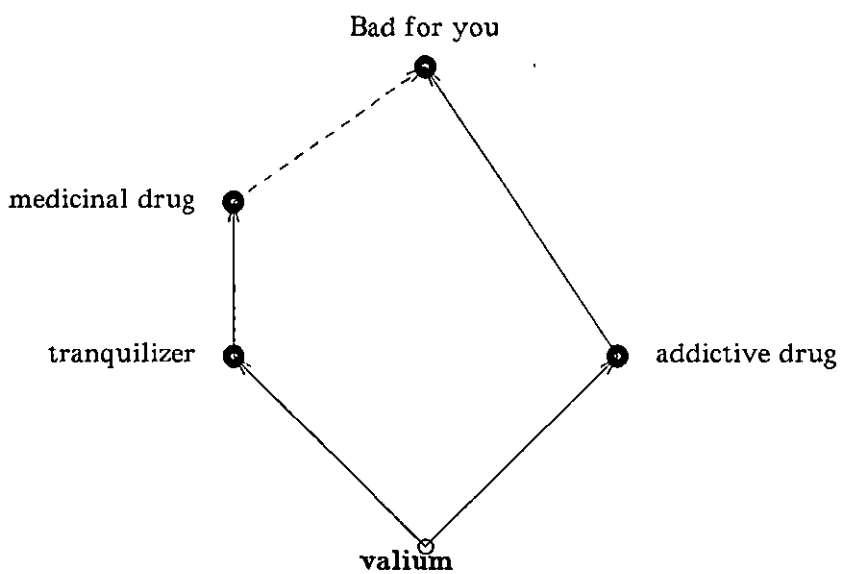


**Figure 2.5 A network with two consistent expansions**

based system is not a contradiction because it is quite conceivable that valium is both of good for you and bad for you depending on the application. If administered medically, then the drug is good for you; if taken such that it becomes addictive, then it is bad for you. Neither of the two conclusions about valium should be lost. The action Touretzky's algorithm takes is merely to know that there is such an ambiguity.

# CHAPTER 3

## Re-assessing the Groundwork

### 3.1. Introduction

The problem of inheritance has now been thoroughly investigated, and the shortcomings of systems which have tried to solve it have been displayed. An inheritance mechanism is doing the job of endowing each item in a hierarchy (or taxonomy) with properties we know it ought to have. As stated at the beginning, given an item which fits somewhere into the network of knowledge that each of us carries around, we instantly know an awful lot more about that item than just its type or name.

What is required is a very thorough analysis of what we are trying to represent in semantic networks. This section shows that there are distinct problems in considering nodes as representing concepts. Even though an examination of the groundwork is required, networks and hierarchies are really quite basic tools for representing knowledge, and a comprehensive philosophical investigation is out of place. Knowledge based systems seek to be pragmatic, and the solution of epistemological questions is not a goal of research. Their philosophies, regarding the status of the elements by which they work, are not of primary concern. The philosophical issues should in no wise be dismissed, but they should be borne

in mind along with the main concern of functionality and usability. Brachman [Bra77, Bra79, BrS85] has commented on the idea of considering nodes as representing bundles of properties. Brachman does not consider cancellation essential, and actually went ahead to build a system (with Schmolze) in which he explicitly excluded cancellation on philosophical grounds [BrS85]. In this system, properties were seen as necessary and not as default.

In the sixties, researchers assumed that the nodes in their systems represented concepts. This was easy to believe since they could label the nodes anything they chose, and they were seduced into believing that the node actually represented whatever was suggested by the node name. Similarly, the links between the nodes seemed to be quite clearly devices to join one concept to another. But by the mid-seventies, it was realised that networks had problems which were not realised or alluded to. One of these problems was inheritance. Doubt was also cast on what it was that links and nodes represented. An influential paper by Brachman [Bra83] highlighted some of these difficulties, although his statement of them is not always as clear and precise as it might be.

### 3.2. Concepts vs. a bundle of properties

The link that was prevalent in early systems was the IS-A link, and one natural property of this link is that it facilitates inheritance throughout the network structure. The inheritance hierarchy was an implicit part of the system, and could not be removed without the whole reasoning system coming down

with it. However, it was not obvious what IS-A *meant*, and this is what caused difficulty in writing systems to represent knowledge. Each system faced the controversy between what the links were actually chosen to represent, and what they ought to represent. Even though at first glance, all the early systems appear to be the same, the question of semantics cast doubts and showed up difficulties. If the number and type of nodes and links was widespread, then so was the number and type of inheritance mechanisms. The apposite question here is, what was being inherited, and what did the inheritance mechanism do?

Brachman spends much of the rest of his article [Bra83] discussing defaults and inheritance. He wants to claim that networks based on standard IS-A links do not function properly, and cancellation is without meaning:

> The more or less standard use of IS-A—as an indicator of default information—brings with it some potentially serious problems. One cannot use a network based on it to represent complex concepts, and the notion of cancellation that follows from it can wreak havoc with world knowledge. The tight association between inheritance and IS-A serves only to further confuse matters. Only by placing inheritance in perspective—it is an implementation issue, not an expressive one—can we clarify what the aims of IS-A really are [Bra83].

Unfortunately, there is little by way of clarification as the paper develops. However, his paper is a production of the era in which confusion was prevalent in the world of semantic networks. Brachman is right if we take a particular view of inheritance systems, but wrong because this view is not the one in which most inheritance reasoners are now built, and therefore his criticisms go wide of the mark for today's systems.

Nodes can be grouped into two types: individuals and generics. All the leaf nodes in a network are individuals. Individual nodes must be treated separately from generic nodes. They are not to be treated either as a concept or as a bundle of properties, although certainly, and individual node may have properties attached. The links therefore connect either two generics or an individual and a generic. The meaning of the links changes according to what the nodes are supposed to represent. Brachman's main criticism of inheritance reasoners is that the nodes are often *labelled* descriptively without bearing the meaning described by the label. It is not reasonable to assume that just because a node is called "soap box" that this is what it actually means. Nor can we suppose that any meaning the node does have is derived from either "soap" or "box." Thus, the labels we choose to represent some concept are not relevant to the meaning of the node itself.

So what do the nodes represent? It seems clear now that nodes in inheritance reasoners do not represent concepts. To represent a concept such as "water" completely, the defining properties of water would probably have to be known, although this is a polemical issue. Such philosophical problems do not override the importance of designing a system whose epistemology is functional and pragmatic. Specifying the essential properties of something is a notoriously difficult issue. The essential, defining, necessary characteristics of water seem easy to define, but they are not. Whichever definition we come up with, for example, "Water is $H_2O$," we can always imagine something in some possible world which

is exactly like water, but which does not have a chemical structure of $H_2O$. Some have argued that since we cannot find the defining properties of something, then we should not identify concepts by such definitions, but should leave the concept ill-defined. Thus, when viewed as concepts, nodes are subject to a whole range of philosophical worries. Nodes seem to be best thought of as a point to hold all the properties pertinent to some particular concept.

If we take Brachman's point about the labels, then a node which holds the properties pertinent to a soap-box might just as well be called TH1599. But this point about the labels only applies if we are thinking of an abstract set of nodes upon which no semantic structure has been imposed. This, however, is not the case. The nodes can be assigned semantic meanings just as one wishes. For example, an inheritance *hierarchy* is a structure which ascribes meaning to the nodes depending on their place in the network.

Nodes as concepts must therefore be abandoned in favour of nodes as holding points for properties. Further, what we decide to call the node is unimportant, but it does gain significance and meaning from its position in a hierarchy, and from whatever the system designer chooses to ascribe to it. Originally, nodes would have been seen as asserting universal statements, but because these properties could not be cancelled, nodes were seen as stating defaults, which are cancellable. IS-A is most often taken to be a default mechanism. Defaults are assumed to be true as long as they are not explicitly cancelled, and are therefore much more useful than universals which could not be excepted.

Considering nodes as holding points for a bundle of default properties is important because of the "strictly one-way nature and cancellability of the properties expressed by the default IS-A's [Bra83]." For example, if sample G45 is quartz, then it has all those properties typical of quartz. What Brachman means by the "strictly one-way nature" of IS-A links is that if we find something which has all the properties typical of quartz, perhaps held at a certain node called "quartz," we cannot assume that it is quartz. Just as if we find something that has all the properties of a wasp, we cannot assume it is a wasp: it might be a very unusual bee.

The author is in agreement with this important point, but whilst talking about the default interpretation of IS-A Brachman moves rather swiftly from making the point above, about nodes being holding points for bundles of default properties, to his criticism of the scheme. His criticism does not go through because he moves from considering nodes which have acquired some semantics, to a network in which there is only syntax. The criticism is that defaults cannot represent the most simple of composite concepts. The example he gives is that of trying to represent "elephant with blue eyes." He says that the best we can do is assert that an Elephant-With-Blue-Eyes typically has blue eyes. But there is confusion between thinking of Elephant-With-Blue-Eyes as being the concept of such an elephant, and as an arbitrary label. Considering the label as arbitrary, then we have no right to say anything about the typical properties at all. In a hierarchy where the node actually does represent a class of blue-eyed elephants,

all the elephant properties would be inherited from the superclass, Elephant, and there would also be an additional property of having blue eyes. But now, the property of blue eyes takes no rôle other than that of all the other elephant properties, inherited from above. Assuming the semantics assures that no elephant without blue eyes would ever be asserted beneath the Elephant-With-Blue-Eyes node, we are saying much more than that such an elephant *typically* has blue eyes.

Brachman goes on to say that a default-based network can make assertions but could never draw any conclusions. "Without being explicitly told," says Brachman, "the system would not know for sure that an elephant with blue eyes *was* an elephant." This is correct if we assume that the nodes have no meaning, but not true otherwise. Since networks nowadays have semantic structures imposed on them, this comment is not relevant.

Considering cancellation, Brachman switches from considering networks where the node names have no meanings to those whose nodes do have meaning, and whose names are therefore meaningful, although these switches are not explicitly stated. He considers a case in which we except the cardinality of legs in elephants (four) and make it three. Then we make an exception to the three-legged elephant, and change it back to four. Now he claims that we could go on cancelling ad infinitum. But this is surely wrong, for if we change the cardinality from four to three to four we do not get three nodes but two, because the last change would be an attempt to create a node already in the network.

Apart from clearing up the confusion surrounding links and nodes, this section showed the importance of specifying what it is that a node represents. The view of a node as concept has largely disappeared, being fraught with difficulties, and the view of the node as a place holder for a group of default properties is being adopted. But the node will only become useful in a network if it is meaningful. Only when the node has been given a specific semantic rôle will inheritance be meaningful. Given that we are committed to representing exceptions, the problem remains as to how this should be done. We want to avoid Brachman's difficulties of representing certain properties, writing out the list with cancellations, and then writing the old list again in another place. The computationally complex addition to the bundle of properties nodes is to hold a list of exceptions at those nodes which have wanted to cancel some property. This way, if Clyde has only three legs, we can still discover that he is an elephant without ever reading the list of exceptions. Only when one of the inherited properties from a superclass is explicitly cancelled do we need to search the exceptions list for an alternative value.

### 3.3. A fresh look at concepts

Early work in knowledge representation lead to the belief that some nodes should be thought of as representing concepts. Further, these nodes are best seen as representing default information which can be inherited by particular individuals when no specific information is known or deducible. But, if the IS-A link is

thought of as a default mechanism, then nodes cannot be thought of as representing concepts. This is because contradictory concepts could not be detected. Rather, nodes must be thought of as holding points for a bundle of properties, because of the strictly one-way nature and cancellability of the properties expressed by the default IS-A's. Thus, *if* Sam is a wasp, then he has properties typical of wasps.

Below is described why concepts were seen as essential, and how early in the development of knowledge representation systems inheritance was recognised as a problem. Brachman calls for an analysis of the fundamentals and foundations of semantic networks [Bra77]. He asks what nodes are for. Nodes can be facts, events, things, sets or concepts. The purpose of nodes is to store information about the world. This information is associated by means of links and nodes are places where knowledge about particular things is stored. There can be object nodes, which represent particular individuals; factual assertions nodes, such as X > Y; and event nodes, such as X hit Y. Nodes are often used to represent groupings of particular things, that is to say, classes of individuals. Any class of individuals can have subclasses. So, for example, the class of *canines* would have the subclasses, *foxes*, *wolves*, and *dogs*.

What Brachman tries to show is that the IS-A link does more than just represent class membership. Class nodes seem to capture *what it means* to be a member of the corresponding class. If we use nodes to represent concepts, then we need to define *what it means to be something* in terms of a group of links that

are attached to a node. The problem is that we cannot do this until we understand what a concept is. This is a major concern in philosophy of knowledge, and the intricacies of the arguments are beyond the scope of this thesis. One view of concepts is that they are defined by bundles of properties. A concept node is meant to represent the general nature of a class of individuals. By expressing all and only those qualities which make something a member of the class, the node is describing all of the potential instances of the concept. Concept nodes will of course have attribute links sprouting from them to represent generalisations from individual nodes.

There are problems with having attribute links attached to concept nodes. First, the attachment of a property to a concept node is usually expressed in the same manner as the indication of the class membership relation. Second, property-asserting links at concept nodes indicate something about the members of the class rather than the class itself. For example, when we say that "Canaries are yellow," we do not mean that the concept of canaries is yellow. Thirdly, in describing potential class members, attribute links may indicate a particular value that holds for every member of the class. More usually however, they just describe the value rather than give it. Two problems stem from this.

(a) The assertion that an attribute has a value in every instance is indistinguishable from the assertion that a particular individual has the value. The former inserts something implicitly about many individuals whilst the latter is asserting something explicitly about a particular one.

(b) When describing what the value of a particular attribute can look like in instances of a concept, an attribute link and its value serve to delineate the class of legal values.

Brachman also reveals some of his worries about inheritance, which are of particular interest to me. He says that we feel that a concept captures what it means to be a particular kind of thing. However, is not a concept much more indefinite than this? It can be quite a "fuzzy" idea, or a very well structured one. However, if concept nodes are to be used in semantic nets, then they must pass on the concept, taken from its node, to particular instances by means of a link. Brachman thinks that the concept is synonymous with *definition*. Here, it seems, is the beginning of the trouble with concepts, for a concept is more than a definition. If it were only a definition, then why would we need the word 'concept'? Nevertheless, Brachman's notion of inheritance is right. *Something* must be passed on from the node which contains a bundle of properties, which somehow capture the idea of a class of individuals, to a particular individual. For example, if Joey is an instance of the concept of a canary, then we expect Joey to inherit a set of properties which are normally attributable to canaries.

This inheritance of properties is uncomfortable because it is *implicit* in semantic networks. It just happens without having a mechanism for making it happen explicitly. There is considerable obscurity when we think about all the things that are going on in this implicit inheritance. Properties that are asserted at concept nodes to have particular values, have those values directly inherited

by instances; descriptions of potential attribute values are instantiated; and special links such as X is an *INSTANCE-OF* Y are not passed on at all. Thus the notation is obscure.

Brachman goes on to define a set of primitives which he considers will clear up this notational difficulty. He says that we must not force knowledge of the world directly into a simplistic node-plus-link system. Instead we must use a set of epistemological primitives as a language in which the parts and features of concepts and instances can be specified in a consistent and extensible way. He puts forward a mechanism for describing the potential instances of a concept. We must explicitly separate value-describing and value-giving.

# CHAPTER 4

## Solutions to the Problem of Exceptions

### 4.1. Introduction

Touretzky "presents a formal mathematical theory of inheritance with exceptions and shows how to correct the flaws in existing inheritance systems [Tou86]." The other contribution of his thesis is a theory of inheritance applied "to the formal analysis of a massively parallel marker propagation machine, of which the most well known example is Fahlman's NETL machine [Fah79, Tou86]."

Touretzky's doctoral thesis begins by reviewing notable inheritance systems and stating the problems of exceptions and multiple inheritance from which they suffered. His own solution to these difficulties is the inferential distance ordering algorithm. This algorithm is an improvement over the previous ones employed by NETL and FRL. For example, when NETL is faced with a contradiction, it resolves that contradiction by making reference to some irrelevant feature of its marker propagation algorithm. This is unacceptable, and Touretzky rightly criticises it as such, seeking for a better solution with his inferential distance ordering algorithm.

Having pointed out the problems that exist in earlier inheritance systems and having expounded an inheritance algorithm which does not falter under the trials of multiple inheritance, contradictions, redundancy and exceptions, he moves on to present a generic inheritance system using the inferential distance ordering algorithm. He provides a mathematically sound basis for inheritance, which is an extremely important contribution to the subject. Assertions to be represented in the system fall into six groups: positive, negative, or a "cannot decide" state, for individuals and then for groups. All these assertions can be represented in non-monotonic logic or as links in an inheritance graph. From these initial observations, the theory of inheritance is built up, and the new theory does not suffer from any of the problems which plagued early systems.

Thus Touretzky offers an informal account of what the inferential distance ordering algorithm is, and then goes on to define it in rigorous mathematical terms. There follows a couple of chapters developing "a mathematical semantics for inheritance networks in terms of constructable sublattices." This theory is introduced to explain an efficient implementation of the inferential distance ordering algorithm; the lattice notation can be used to describe the parallel marker propagation machine derived from Fahlman's NETL [Fah79].

### 4.2. Problems with Touretzky's work

In the course of his thesis, Touretzky has cause to use many examples. One general criticism is that all these examples fail to consider the low-level imple-

mentation details involved. He never gives an account of the low-level working of the algorithm, or how the knowledge might have to be stored in order for a real system to function. For example, the hierarchy in Figure 4.1 looks acceptable at first glance, but when we start to analyse how it might actually appear in a real system, we realise that there are several difficulties with it.

The organisation of the hierarchy is very unnatural. A taxonomy of the animal kingdom in which mammal appears below quadruped is most unusual. Both biped and quadruped are employed as nodes in the network but no hints are given as to what the nodes represent. Brachman's thoughts [Bra77, Bra79, Bra83]

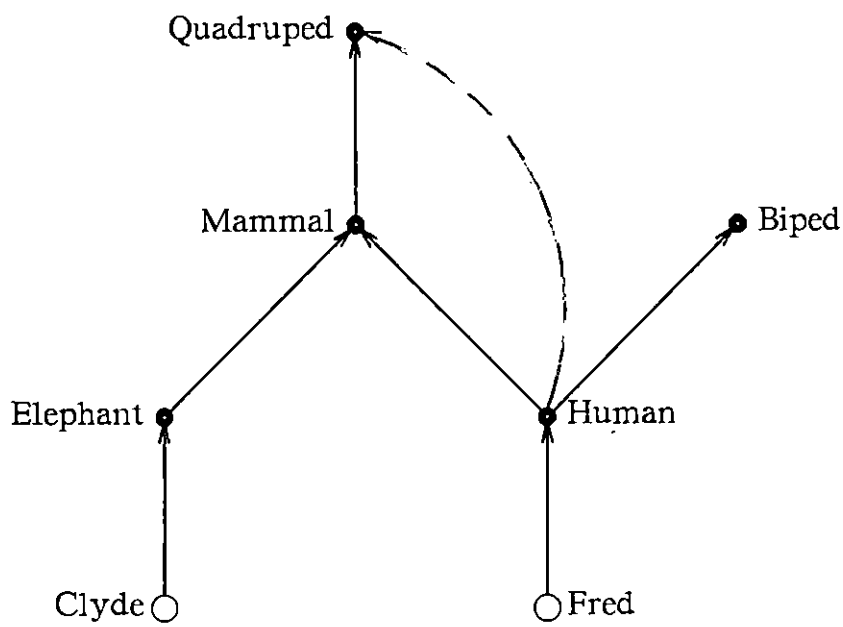


**Figure 4.1 An example network from Touretzky's thesis**

have shown that it is important to know what the node is to represent— whether it be a concept or a holding place. Notice particularly that in Figure 4.1, the Quadruped and Biped nodes are used only to hold properties—the number of legs the creatures below it have. This arrangement is misleading because we now have nodes that appear to represent classes, but are actually only holding places for properties, as well as a hierarchy which is not natural or clear.

How else might the situation above be represented? If the representation can be kept simple, then the inheritance mechanism will also be simple. We do not need a class for Quadruped or Biped, as all these nodes are used for is to keep a count of the number legs of the nodes below them. It is surely more appropriate to have the number of legs of a thing as a property of the thing, and not as the means of partitioning a hierarchy. Thus, one of the properties of the node Mammal would be that mammals have four legs; others would be viviparous, warm-blooded, and so on. By making decisions about where to store properties, we can make a network more readable and simpler, rather than obfuscating the meaning of the relationships between the things represented by the nodes.

Part of what is important about inheritance is choosing the right hierarchical structure and classification of real world objects and concepts. There is, of course, no single "correct" hierarchy with which to represent the world. However, if the aim was to construct a general purpose reasoner, then there would have to be a process whereby the hierarchy could be rearranged or reordered; no one organisation will suit all needs, and talk of natural hierarchy has less force.

Herein, I am considering a hierarchy for a limited domain for which there will be a natural hierarchy and unnatural ones. If an unnatural classification is chosen, then the representation becomes more complex and therefore so does the inheritance of properties. In choosing some representation of the world, one does not have to be bogged down in trying to decide which properties are contingent and which are necessary because we are not trying to represent concepts. The nodes should be given flexibility by considering them as holding points for a real world object. Nodes in a network should capture the state of knowledge we have about the object in the world. The nodes should accurately represent what we know about something and the system should not make incorrect deductions based on incomplete knowledge if that knowledge is available. Although some caution is necessary when considering inheritance of most properties, the inheritance of *some* properties needs to occur without hindrance. For example, every atom has an atomic weight and every man is mortal. These necessary truths should not be cancelled under any circumstance in which we are trying to represent the real world.

Let us move from the point that the whole hierarchy has to be well structured to the consideration of the representation of the nodes themselves. In his thesis Touretzky developed a rigorous theory of inheritance. When he wrote, inheritance systems were being developed, but the focus of research in the field was not representation. Even allowing for the tenor at the time it is fair to say that Touretzky did not consider representation enough in his work. It has since

been realised that representation of knowledge plays an important rôle in determining what that knowledge means. Schubert has suggested a new way to represent the knowledge we have about an object by [CoS80, SGC79]; it should be stored on what he calls topical access hierarchies.

One way to store the properties in a network would be to have them all radiating from the node, and at the same level. A frequent task is to scan a node to see if a particular property is present. For example, if the node in Figure 4.2 were to represent Building, then all the properties of a building, such as its size, its function, and the number of windows it has would be sprinkled around the node in an arbitrary fashion. Suppose a particular building, say a church, appears below the building node. We may now ask the question, how many walls does it have? If explicit information does not exist at the object node, we can look back

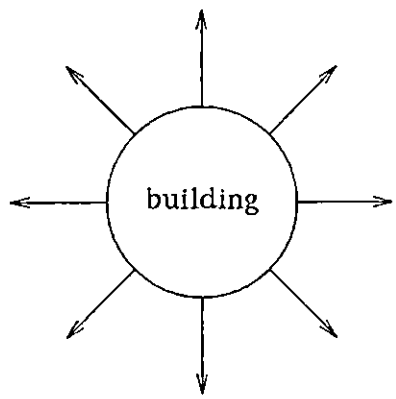


**Figure 4.2 Properties at one level emanating from a node**

up the hierarchy to the Building node, and discover there that, typically, a building has four walls, so typically, a church has four walls. But if we do not know where to find the "number of walls" property amidst all the other properties, then the search time to answer this question would be extremely long.

Schubert has done a great deal of research in trying to organise properties of a node, gradually developing his topical access hierarchies. These hierarchies serve to classify properties to such an extent that they form a tree. In this organisation, we arrange the properties of a node according to certain topics. Schubert thinks that about twelve topics will capture all aspects of the world for which we might want properties. His classifications include physical quality, mental quality, and behaviour. These topics are then subdivided more and more finely until the required property is found. Figure 4.3 shows part of the physical quality hierarchy. This structuring of properties is a considerable improvement over the last attempt, which simply added properties arbitrarily to a node. Here we have a mechanism by which we can store and access properties quickly and minimise the costs for search and retrieval. If mechanisms exist to retrieve properties efficiently, then much headway has been made in making inheritance pragmatic. With the assistance of the topical access hierarchy, perhaps there is a way of simplifying all that Touretzky has done with exceptions. Certainly by representing his examples in a new way, many of the difficulties that he illustrates are less of a problem.

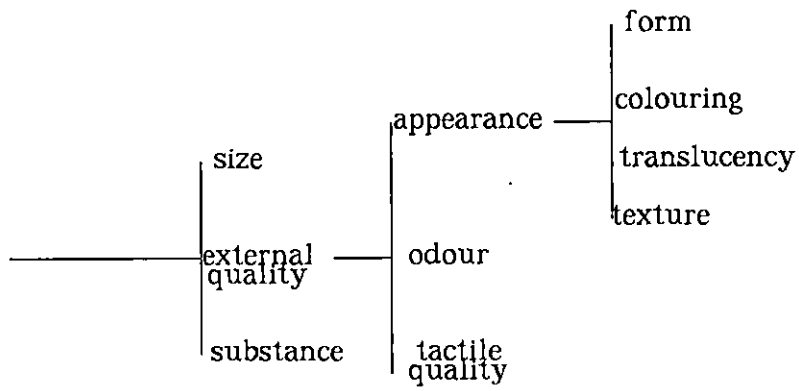


**Figure 4.3 Part of Schubert's physical quality hierarchy**

## 4.3. Where to hold exceptions

In all his examples, Touretzky does not address the difficulty of where, in some scheme, the exceptions would be held. His examples show crossed arrows to indicate that an exception is being made, but this can only be diagrammatic. Somehow, we must take account of the exceptions and hold them somewhere in the network. The most obvious place to hold them might seem to be at the node which makes the exception. If some individual does not want to inherit the property grey from a higher node, then a note is made at the individual node that this is not the case. One of the disadvantages of this scheme is that the exceptions are spread throughout the network and cannot be seen together. For example, Figure 4.4 shows a part of a hierarchy describing clocks. The exceptions to the

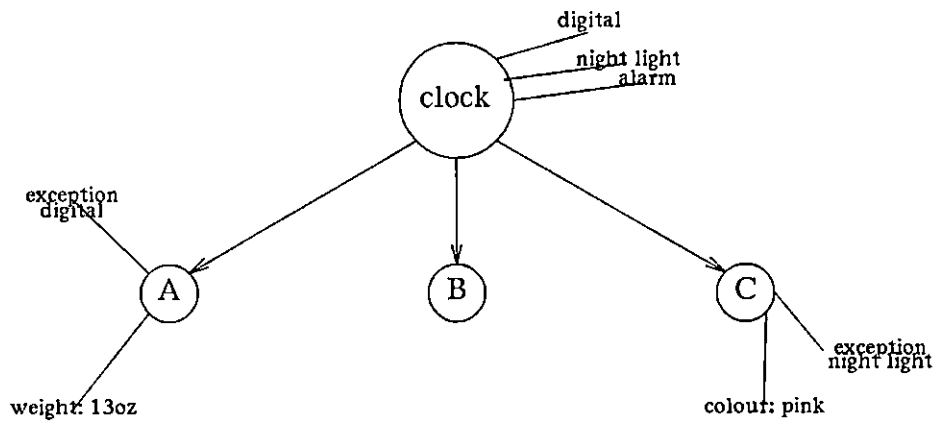


**Figure 4.4 Storing exceptions at the individual node**

clock node appear at the individual nodes. Clock A is not digital, and therefore makes a note of the fact that it does not want to inherit the digital property from its super node. Similarly, clock C takes exception to the night-light property listed at the clock node. Because the exceptions are held at the level of the individuals, more effort will be required when searching through the network to discover which properties a particular individual has since search effort is centred on the individuals, lower down the hierarchy.

The author proposes an alternative scheme, which offers more advantages, which is to hold a list of exceptions at the node a minimal distance above the one which makes the exception. The distance is measured in terms of the number of links. In manipulating the network it may well be convenient to access all the

exceptions at once. This scheme is an improvement over storing the exceptions with the individual which makes the exception because it is more efficient in terms of storage and in terms of search. The exception for which there are many individuals can be stored in one place, but even so a record must be kept of all the individuals making that exception. More importantly, the inherited exceptions are indicated through inheritance mechanisms and not through search. This is a major advantage for it will make search faster and the inheritance mechanism more transparent.

Wherever we choose to hold the exceptions, there is going to be some kind of search involved whenever the inheritance system is required to answer questions. If a question makes reference to a particular property of either an individual or a class, then a search must be made by the inheritance mechanism to locate the property. If the property is inherited, then some extra work must be done to locate possible exceptions. To save scattering the exceptions to a property all over the network, it is more efficient to hold them at the node a minimal distance above the one making the exception. Associated with each node is a topical access hierarchy of properties and the individuals or classes which wish to take exception to a particular property in that hierarchy are held on an exceptions list accompanying it.

Consider Figure 4.5 in which Touretzky's diagram is given first, and then rearranged to demonstrate an alternative. Touretzky's diagram shows some of the difficulties already discussed. First, his diagram reveals a hierarchy which is

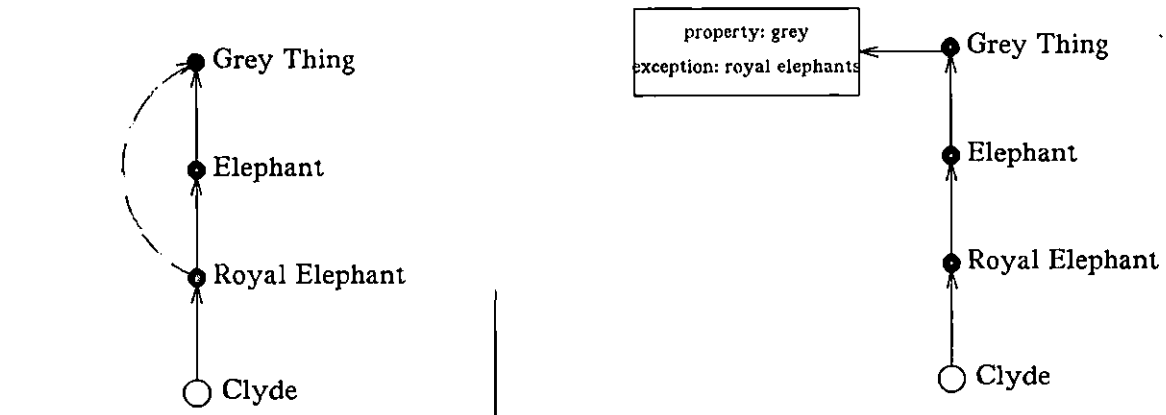


**Figure 4.5 Where exceptions can be maintained**

unnatural, in that the most generic node is Grey Thing. Second, there is no consideration given to how the exception that royal elephants are not grey should be held. In the author's rearrangement, the order of the hierarchy has been maintained for the sake of comparison, although this need not have been so. The node Grey Thing could be dispensed with entirely, and the property grey, the only property belonging to grey thing, could be moved down to be a property of elephant. Royal elephants are not grey, and therefore should not inherit from elephant that property of greyness. This exception is held with the property of greyness at the Grey Thing node, and royal elephant appears on the exceptions list.

Resulting from this chapter are three main points. Touretzky's work shows a lack of reference to an underlying representation. This makes his thesis very

abstract and less pragmatic. His examples are all diagrammatic, and seem to be ideal for demonstrating certain difficulties. However, certain problems simplify when a realistic representation is used. Secondly, his examples are often very unnatural hierarchies. The examples which he uses in his thesis appear contrived and impractical. This surely indicates a lack of consideration for a real representation. Finally, a new solution to the problem of exceptions has been offered.

# CHAPTER 5

# Conclusions

## 5.1. Summary

The first chapter showed where the problem of inheritance is located within artificial intelligence. The problem of inheritance was then stated in an informal way. The second chapter presented a historical summary of the problem, and expanded it to a precise account, illustrating under what circumstances and in what form the problem arises. From the very beginning in knowledge representation, inheritance has played a part, although the full significance of that rôle was not always realised. Once a problem was identified, some researchers tried to find solutions, and these have been analysed. All these previous solutions, some of which are quite recent, contributed to Touretzky's work [Tou86] in which he wrought a mathematical theory of inheritance for the first time. Touretzky's solutions to the problem in its various manifestations were examined and commented upon in Chapters 2, 3 and 4.

Having taken the reader on a journey from the perimeters of artificial intelligence into the involuted details of the problem as seen in various knowledge-based systems attempting to tackle inheritance, the journey turned towards a reexamination of the fundamentals. As mentioned above, Brachman has directed

computer scientists to the philosophical foundations of their research, and in the same vein, this thesis expounds the groundwork of knowledge representation. From this investigation came the conclusion that precisely what is being represented is crucial to understanding inheritance. The author agrees with Brachman that nodes must be thought of as a bundle of properties.

The final part of this thesis revealed some deficiencies in Touretzky's work [Tou86]. An efficient solution to the problem of exceptions is proposed.

## 5.2. Conclusions

Drawing together the divergent lines of research is perhaps the most important task accomplished herein. There is a surprising paucity of research upon which to draw even though it covers a broad area. It is surprising because the various aspects of inheritance seem so central to knowledge representation systems. An overall statement of the problem and the proposed solutions thus far have been presented for the first time. There are many papers which deal with a very restricted aspect of the problem, and many which examine the problem in relation to one particular system, but none that cover the whole problem.

Many of the difficulties that have arisen, not just in the field of inheritance but also in other areas of knowledge representation, have been caused because of confusion at the level of fundamentals. Brachman has published extensively on precisely this topic, concentrating particularly on concepts [Bra77, Bra79, Bra83]. The author was inspired to go back to the roots of the inheritance problem and

take a fresh look at what was being represented in semantic networks. The third chapter of the thesis has provided alternative perspectives on the problem, enabling other researchers in this field to continue to explore inheritance.

### 5.3. Further directions for study

Inheritance is gradually acquiring the attention it deserves considering its centrality to knowledge representation. It is surprising to see how much more knowledge has been gained through the efforts of a great many researchers who have found that the problem of inheritance blocks the way to a solution of a contingent problem. Touretzky seems to have provided an adequate solution to the problem of multiple inheritance which plagued earlier systems. His inferential distance ordering algorithm has yet to be tried and tested extensively, but it has been implemented on one machine by him as part of his thesis. One direction for further study would be the implementation of this algorithm in several different knowledge-based systems. The proposal of further efficient solutions to the problem of exceptions is a major area of research for the future.

Because the problem of inheritance is so deeply rooted in our thinking, it appears to defy an all-encompassing solution. Once it is broken down however, some aspects of the problem lend themselves to fairly easy solution. The idea of having individuals and classes inherit properties from other classes is really quite a simple one, and tends to be lost when the details of the problem are concentrated upon. Because the basic idea is so simple and so essential in the real world,

it is important that it is solved. The breakthrough in the solution to the inheritance problem will come when a system is constructed which is able to represent knowledge in such a way that it can cope with facts and the changes that occur over time in a way which is similar to the behaviour of humans.

The proposals made in the last chapter regarding exceptions could be implemented in an inheritance system. It would be beneficial to combine those proposals with Touretzky's solution to the problem of multiple inheritance—his inferential distance ordering algorithm.

## References

[BoW77]

D. G. Bobrow and T. Winograd, "An Overview of KRL, a Knowledge Representation Language," *Cognitive Science 1*, 1 (1977), 3-46.

[BBB70]

"Mixed Initiative Man-Computer Instructional Dialogues," in *BBN Report Number 1971*, Bolt Beranek & Newman, Cambridge, Massachusetts, 1970.

[Bra77]

R. J. Brachman, "What's In a Concept: Structural Foundations for Semantic Networks," *International Journal of Man-machine Studies 9*, (1977), 127-152.

[Bra79]

R. J. Brachman, "On The Epistemological Status of Semantic Networks," in *Associative Networks*, N. Findler (ed.), Academic Press, New York, 1979, 3-50.

[Bra83]

R. J. Brachman, "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks," *Computer 16*, 10 (October, 1983), 30-36.

[BrS85]

R. J. Brachman and J. G. Schmolze, "An Overview of KL-ONE Knowledge Representation System," *Cognitive Science 9*, 2 (1985), 171-216.

[Car70]

J. R. Carbonell, "AI in CAI: An Artificial Intelligence Approach to Computer-Aided Instruction," *IEEE Transactions on Man-Machine Systems MMS-11*, 4 (1970), 190-202.

[CoQ69]

A. M. Collins and M. R. Quillian, "Retrieval Time from Semantic Memory," *Journal of Verbal Learning and Verbal Behaviour 8*, (1969), 240-247.

[CoS80]

A. R. Covington and L. K. Schubert, "Organisation of Modally Embedded Propositions and of Dependent Concepts," *Proceedings of the Third Biennial Conference of the Canadian Society for Computational Studies of Intelligence* , Victoria, B.C., 14-16 May, 1980, 87-94.

[EtR83]

D. W. Etherington and R. Reiter, "On Inheritance Hierarchies with Exceptions," *Proceedings AAAI-83* , 1983, 104-108.

[Fah79]

S. E. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*, MIT press, Cambridge, Massachusetts, 1979.

[FTR81]

S. E. Fahlman, D. S. Touretzky and W. Roggen, "Cancellation in a Parallel Semantic Network," *Proceedings IJCAI-81* , Vancouver, BC, 1981.

[GoR77]

I. P. Goldstein and R. B. Roberts, "NUDGE, A Knowledge-Based Scheduling Program," *Proceedings of the 5th international joint conference on Artificial intelligence* , Cambridge, Mass., 1977, 257-263.

[GoR79]

I. P. Goldstein and B. Roberts, "Using Frames In Scheduling," in *Artificial Intelligence: An MIT Perspective*, P. H. Winston and R. H. Brown (ed.), MIT Press, Cambridge, Mass., 1979, 253-286.

[Min75]

M. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York, 1975.

[Qui66]

M. R. Quillian, "Semantic Memory," in *Report AFCRL-66-189*, Newman, Cambridge, Mass., 1966.

[Qui69]

M. R. Quillian, "The Teachable Language Comprehender: A Simulation program and theory of language," *Communications of the ACM 12*, 8 (1969), 459-476.

[Rei81]

R. Reiter, "A Logic for Default Reasoning," *Artificial Intelligence 13*, 1,2 (1981), 81-132.

[Rei83]

R. Reiter, "Some Representational Issues In Default Reasoning," *Computer and Mathematics with Applications 9*, 1 (1983), 15-27.

[RoG77]

R. B. Roberts and I. P. Goldstein, "The FRL Manual," *AI Memo 409, MIT AI Lab* , Cambridge, Mass., 1977.

[SGC79]

L. K. Schubert, R. G. Goebel and N. J. Cercone, "The Structure and Organisation of a Semantic Net for Comprehension and Inference," in *Associative Networks*, N. Findler (ed.), Academic Press, New York, 1979, 121-175.

[Tou84]

D. S. Touretzky, "Implicit Ordering of Defaults in Inheritance Systems," *Proceedings of AAAI-84* , Austin, Texas, 1984.

[Tou85]

D. S. Touretzky, "Inheritable Relations: A Logical Extension to Inheritable Hierarchies," *Proceedings of the Workshop on Theoretical Approaches to Natural Language Understanding CSCSI/SCEIO*, (1985), 55-60.

[Tou86]

D. S. Touretzky, *The Mathematics of Inheritance Systems*, Morgan Kaufmann Publishers, Los Altos, California, 1986.

[Win75]

T. Winograd, "Frame Representations and the Declarative/Procedural Controversy," in *Representation and Understanding*, D. G. Bobrow and Collins (ed.), Academic Press, New York, 1975, 185-210.

[Woo75]

W. A. Woods, "What's In a Link: Foundations of Semantic Networks," in *Representing and Understanding*, D. G. Bobrow and A. Collins (ed.), Academic Press, 1975, 35-82.

# VITA

*Surname:* Cripps

*Given Names:* Andrew David

*Date of Birth:* 28th January, 1963

*Place of Birth:* Crawley, Sussex, England.

## Educational Institutions Attended

| | |
|---|---|
| *University of Victoria, Victoria, BC* | 1986-1987 |
| *University of Reading, Reading, Berkshire, England* | 1983-1986 |

## Degrees Awarded

| | | |
|---|---|---|
| B.Sc. (Honours) | 1986 | *University of Reading, Reading, Berkshire, England* |
| M.Sc. | 1987 | *University of Victoria, Victoria, B.C.* |

## PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis (the title of which is shown below) to users of the University of Victoria Library, and to make *single copies only* for such users or in response to a request from the library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Aspects of the Inheritance Problem

Author [redacted]

1st September, 1987
Date