Inheritance Comes of Age: Applying Nonmonotonic Techniques to Problems in Industry

Leora Morgenstern
IBM T.J. Watson Research Center
30 Saw Mill River Road, Hawthorne, NY 10532
leora@watson.ibm.com

Abstract

Nonmonotonic reasoning is virtually absent from industry and has been so since its inception; the result is that the field is becoming marginalized within AI. I argue that this is because researchers in the area focus exclusively on commonsense problems which are irrelevant to industry and because few efficient algorithms and/or tools have been developed. A sensible strategy is thus to focus on industry problems and to develop solutions within tractable subtheories of nonmonotonic logic. I examine one of the few examples of nonmonotonic reasoning in industry — inheritance of business rules in the medical insurance domain — and show how the paradigm of inheritance with exceptions can be extended to a broader and more powerful kind of nonmonotonic reasoning. Finally I discuss the underlying lessons that can be generalized to other industry problems.

1 Introduction

Nonmonotonic logic, the formalization of plausible reasoning, is invisible and virtually non-existent in industry. It is in a worse position, in this respect, than most other areas of Artificial Intelligence. It is true that AI researchers have long accustomed themselves to the huge gap between AI hype, which promises great things (e.g., housekeeping robots) and AI reality, which delivers much less (robots that have a hard time collecting tennis balls).

Yet AI as a whole is quite visible in industry and the marketplace. Although AI has delivered less than was anticipated one or two decades ago, there is enough going on: Expert systems are used in medical diagnosis, circuit configuration, and financial applications; dictation systems for restricted domains are on the market. Unfortunately, such examples don't include anything that is based on nonmonotonic reasoning.[1]

The absence of nonmonotonic reasoning from industry may have been small cause for concern in the early eighties, when AI showed endless promise, research money was plentiful, and the field was very young. As we approach the end of the nineties, however, we have reason to worry. Funding has shrunk, and there is little tolerance for research programs that don't promise — and deliver — practical results in the foreseeable future. There is the real danger — if nonmonotonic reasoning and industry continue to inhabit separate worlds — that nonmonotonic reasoning will become marginalized and isolated; that funding for nonmonotonic research will dry up to the point where we are no better off than researchers in mathematics (or worse, philosophy); that as a result the field will shrink, leaving only a few die-hards talking to one another instead of a vibrant research community which is tackling one of the hardest problems in reasoning.

I don't feel happy about writing that last paragraph. These are the sort of gloomy prognostications one is used to hear from people outside the field of nonmonotonic logic. When I've heard these sentiments in the past, I've usually put them down to some combination of *schadenfreude* and the resentment of practitioners toward theorists. This isn't the case here. On the contrary: I'm a member of the nonmonotonic reasoning community, and I'm concerned about the current state of nonmonotonic research. But this is concern rather than pessimism: I believe that we can stop the field from being marginalized, and strengthen nonmonotonic reasoning as a central part of mainstream AI. In order to do so, we must find some way to make nonmonotonic reasoning useful to industry. We need to understand why nonmonotonic reasoning and industry are so far apart and to figure

[1] It might be argued that fussy logic (Zadeh, 1992), which also claims to capture plausible reasoning, has been used in industrial applications. Without commenting on the merits of this argument, we merely note that the fields of fussy logic and nonmonotonic logic are quite separate in both philosophy and community; thus, the presence of fussy logic in industry says little about the field of nonmonotonic reasoning. It is also true that the logic programming language Prolog (Clocksin and Mellish, 1987) is used in industry: while its negation-as-failure mechanism makes it in theory suitable for expressing certain types of nonmonotonic reasoning, Prolog in the commercial world is generally not used to capture nonmonotonicity and thus contributes little toward joining nonmonotonic research and industry.

out how to bridge the gap. We also need to see if there are any examples of nonmonotonic reasoning in industry, and to study these examples for lessons to generalise and mistakes to avoid in the future.

The paper is accordingly structured as follows. Section 2 discusses the reasons underlying the gap between nonmonotonic reasoning and industry and suggests possible strategies to bring these two areas closer together. Section 3 examines in detail an example of a nonmonotonic system developed for industry — specifically, a benefits inquiry system for the insurance industry. The system uses a form of inheritance with exceptions in which logical rules — *well-formed formulae* — are attached to nodes in the inheritance network. The system is able to perform broad nonmonotonic reasoning. We examine the ways in which nonmonotonic techniques provide the system with the necessary expressiveness and reasoning ability. We subsequently consider generalizations of the system, and finally examine the lessons which can be applied in general to joining nonmonotonic reasoning and industry.

2 Analyzing the gap between nonmonotonic reasoning and industry

Nonmonotonic reasoning was first introduced in the late 1970s (McDermott and Doyle, 1980; Reiter, 1980; McCarthy, 1980) in order to formally capture plausible or default reasoning.[2] Plausible reasoning includes reasoning with exceptions, reasoning with default rules (rules that talk about a typical member of a class, rather than all members of a class), reasoning with incomplete information, and jumping to conclusions and retracting such conclusions if they are proved to be wrong. The aim in those early days was to construct a logic that is more powerful than classical first-order logic and to aid in developing programs that could reason more flexibly and fluently than the programs then available. Nonmonotonic logic was supposed to make AI easier. As such, it could have been reasonably expected that nonmonotonic logic would become a tool of software engineers (as is the case, for example, with object-oriented programming). In fact, this has not happened: two decades later, open activity in nonmonotonic research is found only in academia and tolerant research labs.

What went wrong? The answer, in a nutshell, is that nonmonotonic reasoning is nowhere near ready to handle industrial-strength problems. Researchers freely admit this, and have been freely admitting it for the last twenty years. After this length of time, the admission is in itself cause for concern. Much of AI and all of industry is about getting things done. Confession will not save us here: we need to determine why nonmonotonic reasoning hasn't helped get things done. Three reasons come to mind.

(1) Nonmonotonic research has focussed almost exclusively on toy problems of commonsense reasoning. The canonical Tweety problem — inferring that Tweety can fly from the facts that Tweety is a bird and that birds typically fly; and retracting that conclusion upon discovering that Tweety is a penguin — is still one of the benchmark problems that researchers seriously tackle when they develop new nonmonotonic logics or modify old ones.[3]

Indeed, nonmonotonic theories have trouble solving a host of other toy problems such as the well-known Yale Shooting problem (Hanks and McDermott, 1986), which involves predicting that if one loads a gun, waits, and then fires the gun at a turkey, the turkey will die.[4]

It can be argued with a good deal ofjustification that commonsense reasoning is one of the more difficult areas of intelligent behavior for AI to model (Davis, 1990) and that sneering at research on the Tweety and Yale Shooting problems merely reflects a lack of understanding of the difficulties of the underlying issues. It may very well be that toy problems of commonsense reasoning are more difficult than industry problems. Nonetheless, one can hardly be surprised that industry has not jumped to invest in technology based on research that is stymied by the likes of the Yale Shooting problem.

(2) In fact, industry is primarily concerned with problems which have very little to do with commonsense reasoning. Examples include diagnosing bacterial infections, determining where oil is likely to be found, and predicting variations in the stock market. Nonmonotonic researchers typically ignore these problems, preferring instead to work on problems of commonsense reasoning, as discussed above. The result is that these researchers have very little to offer industry. The irony is that one can plausibly argue that non-trivial nonmonotonic reasoning is present in a wide variety of industry problems. For example, virtually any prediction task must be done in the absence of complete information about one's situation, and must use causal rules which have exceptions; this suggests that some form of nonmonotonic reasoning (such as nonmonotonic temporal reasoning) is needed.

Industry seems to offer fertile ground for nonmonotonic researchers. The problem is that industry remains uncharted territory for the nonmonotonic community.

[2] These papers as well as some of the classic papers in the field are collected in (Ginsberg, 1987).

[3] While all existing nonmonotonic logics, as far as I know, can solve the Tweety problem, simple variations on this problem are beyond some well-known nonmonotonic systems. For example, a nonmonotonic reasoning system based on consequence relations (as in Kraus, Lehmann, and Magidor, 1990) cannot infer that Tweety can fly from the facts that Tweety is a robin, robins are birds, and birds typically fly. (The problem is that chaining is in general not permitted.) A relatively simple fix (Geffner, 1990) results in a system that can solve this variant Tweety problem; the point, however, is that it is far from obvious that nonmonotonic systems can solve very simple reasoning problems.

[4] It is assumed that firing a loaded gun at a turkey always results in the death- of the turkey. The difficulty arises in predicting that a gun that was loaded at one moment will remain loaded at the next. Early papers on the Yale Shooting problem can be found in (Ginsberg, 1987); for a recent analysis, see (Morgenstern, 1996c).

The result is that this community has not yet demonstrated that it is capable of solving any industry problems. It is all very well to argue that the seemingly hard problems facing industry are easier to solve than the deceptively simple commonsense problems upon which nonmonotonic research focusses, but this argument must be buttressed with solid solutions to problems in industry.

(3) Nonmonotonic reasoning techniques have not scaled up to industry. Even if the nonmonotonic community were to start working on a problem directly relevant to industry, and to come up with a good solution, nonmonotonic reasoning is crippled by decidability and tractability problems, and a lack of good tools. Specifically, nonmonotonic predicate logic is in general undecidable; even simple classes of propositional nonmonotonic logics are intractable. For example, determining whether a formula is in the extension of a propositional default theory is in general Σ_2^P-complete (Gottlob, 1996).

There are some bright spots in this otherwise dark picture. There are relatively efficient polynomial algorithms for a particular type of nonmonotonic reasoning known as *inheritance with exceptions* (Horty et al., 1990; Stein, 1992).[5] Inheritance with exceptions is the nonmonotonic extension of inheritance, a simple form of reasoning with subclasses and superclasses that dates back to Aristotle and the syllogism (Kneale and Kneale, 1962). This extended form of inheritance allows one to posit exceptions to the general behavior of classes and to reason with those exceptions. It easily handles Tweety-style problems (but cannot handle the Yale Shooting problem unless that problem is reformulated in a somewhat unnatural manner). In addition to work in this area, there are many promising results in the subfield of nonmonotonic reasoning known as logic programming: for example, computation of solutions for theories that are propositional, non-recursive, and in Horn form is $\mathcal{O}(n)$ (Dowling and Gallier, 1984); computation of a unique solution set for *courteous logic programs,* a restricted form of prioritised defaults, is $\mathcal{O}(n^2)$ (Grosof, 1997b).

But even these positive results are weakened by the lack of corresponding industrial-strength tools which implement these algorithms. For example, there are no commercially available tools for inheritance with exceptions, despite the fact that efficient algorithms have been known and published for almost a decade. Anybody who wishes to use the technology in industry must build the code from scratch. In an age and an industry where tools have become a sine qua non, the lack of a good tool can freeze any possibility of using a nonmonotonic technique.[6]

[5]The tractability of Horty et al's algorithms is discussed in (Selman and Levesque, 1993); the complexity depends on the kind of chaining involved in path construction. The version presented in (Horty, 1994, Section 2.1) leads to tractable algorithms. Stein's algorithm is $\mathcal{O}(n^3)$.

[6]This was, indeed, my experience in developing the system described in section 3. Despite the fact that it was clear that standard inheritance would not do the job, and that at least inheritance with exceptions was needed, the fact that tools that performed standard inheritance existed, while tools that performed inheritance with exceptions did not exist, caused many involved with the project to strongly suggest that the existing tool be used. It is a mark of the Dilbertian nature of the software and consulting industry today that using an existing tool to get the job done badly but quickly is considered preferable to building a tool and doing the job slowly but well.

If the field of nonmonotonic logic has remained entirely separate from industry because first, nonmonotonic research and industry focus on very different problems, and second, researchers have not yet developed efficient algorithms and/or tools, the strategy for integrating nonmonotonic reasoning with industry becomes clear:

First, researchers in nonmonotonic logic should familiarise themselves with problems in industry, select a set in which nonmonotonic reasoning appears to be important, and focus on those problems in their research. Second, researchers ought to actively design efficient algorithms for the tractable portions of nonmonotonic theories, and develop industrial-strength tools.

Ideally, these endeavors should be carried out simultaneously. That is, as solutions to nonmonotonic problems in industry are found, tools to implement these solutions should be developed. That is not essential, however; what is important is that both tasks get done.

At this point, there are at best a handful of industry problems that have been solved using nonmonotonic techniques. The remainder of this paper will discuss one such problem and its solution. We will outline the problem, explain why nonmonotonic reasoning is necessary, present the nonmonotonic techniques used, and suggest how this method could be generalized to other problems in industry.

3 The Case in Point: Inheritance and inheriting rules in the medical insurance domain

[7]

3.1 The Problem: Benefits Inquiry

Problem Description

Benefits inquiry is the process of querying an insurance company to determine one's benefits. In the medical insurance industry, customers may wish to know if a particular procedure is covered, as well as the specific rules that limit coverage. Examples are:

Will my son's tonsillectomy be covered? Can it be performed in an inpatient facility?

How many days can I stay in the hospital after a standard delivery?

[7] A more detailed description of the knowledge structure and algorithms summarised in this paper can be found in (Morgenstern, 1996a) and (Morgenstern and Singh, 1997).

[8]In this paper, as well as in (Morgenstern and Singh, 1997), the term benefits inquiry also refers to the process performed by insurance company employees in answering such queries.

Benefits inquiry occurs frequently in the medical insurance industry and has become increasingly complex: medical insurance companies today may have thousands of insurance products, each of which contains a myriad of services and regulations which change frequently. The vast amount of changing information is difficult to keep up with. In addition, there are many rules that have exceptions, and exceptions can be nested. For example, physical therapy is generally limited to twenty visits per year, unless more visits are ordered in writing by a physician, but spinal manipulation, a type of physical therapy, has a more generous limit (around thirty visits). The importance of exceptions suggests that some form of nonmonotonic research would be useful.

Several years ago, I was asked to develop an expert system for benefits inquiry. (This was part of a comprehensive consulting engagement between IBM Research and a large medical insurance corporation to update major portions of their information management system.) The primary goal of the expert system was to aid customer service representatives (CSRs), the insurance company employees who answer customers' questions about their benefits.

What had been done

Customer service representatives were at that time using a "desktop" text-based system. Information was divided into subject areas such as preventive care, immunization, and drugs; each subject area was associated with a piece of text, roughly the amount that would fit on a screen, highlighting salient pieces of information. For example, the screen on preventive care listed the types of preventive care available, such as routine physicals and immunizations, as well as coverage rates and allowed frequency of services. A topic mentioned in one screen could itself have a full screen devoted to it; thus, for example, immunization, mentioned in preventive care, was the subject of one of the screens.

Although the desktop system allowed rudimentary search and indexing, it was deficient in several respects:

First, only a small amount of domain knowledge was encoded in the system. The amount of information that could be contained was strictly limited: a too-long menu would prove unwieldy to the CSRs; on the other hand, if the chunk of information associated with a menu topic was too large, it would not fit on one or even several screens.

Second, the system did not make explicit the interconnection between subject areas. For example, nothing in the system indicated a connection between the screens on immunization and preventive care. The CSR had to reason that the schedule rate for preventive care probably applied to immunizations.

Third, the system was difficult to update, and updates had to be performed manually. This could be especially troublesome when screens were interconnected. For example, if both the preventive care and immunization screens have schedule rate information and this schedule rate changes, the individual modifying the system must make changes on both screens.

Fourth, due perhaps to the constraints just mentioned, the system was intended to handle only the most frequently asked questions.

Project Goals

We aimed to develop an expert system that supports benefits inquiry but avoids the drawbacks of the text-based system. In particular, this meant a system that allows questions at varying levels of granularity, gives unambiguous answers, allows representation of large amounts of material and navigation around a large information space, supports connections among related topics, and supports easy updates and modifications.

The ability to modify is important because products change so frequently. Thus, the system had to be usable not only by CSRs, but also by *policy modifiers* (PMs), the insurance company employees responsible for making changes within a particular insurance product.

3.2 Why Inheritance with Exceptions is Useful

Much of the information about medical services is taxonomic in nature. For example, Spinal Manipulation is a type of Physical Therapy; Physical Therapy, Speech Therapy, and Occupational Therapy are all types of Therapy. Coverage and accompanying restrictions are to a large extent inherited along taxonomic lines: Physical Therapy is covered, for example, because it is a subtype of Therapy, and Therapy is covered. On the other hand, there are exceptions: even though Drugs are covered by the Drugs Benefit, and OTC (over-the-counter) Drugs are a subclass of Drugs, OTC Drugs are not covered by the Drugs Benefit. Thus, we could not use a standard inheritance network such as KL-ONE (Schmolze and Lipkis, 1983) or K-REP (Mays et al., 1991). We needed at least the expressive power of an inheritance network with exceptions (Horty et al., 1990; Stein, 1992).

Indeed, the structure needed to represent the organization of medical services and benefits is not purely taxonomic, since certain services have multiple supertypes. For example, Genetic Testing is both a subtype of Diagnostic Services and of Family Planning Services. Thus the structure is a dag (directed acyclic graph) rather than a tree. [9]

[9] In fact, a dag is needed to represent all cases of inheritance with exceptions. Formally, multiple inheritance arises when there is an undefeated path from x to y, an undefeated path from x to z, and $y \neq z$ (see below for definitions of these terms). Inheritance networks in the literature have traditionally considered multiple inheritance only when these multiple paths have been initial segments of conflicting paths, as is the case in Figure 1, where the positive path from OTC to Drugs and the negative path from OTC to Services Covered by Drugs Benefit are initial segments of conflicting paths. Here, we will also be interested in *non-conflicting path* multiple inheritance; cases of multiple paths that are *not* initial segments of conflicting paths. For example, in Figure 1 there are non-conflicting paths between Insulin Syringes and

Inheritance with exceptions is all that is needed to determine which medical services are covered by which benefit. Indeed, the first cut at the benefits inquiry system used inheritance with exceptions to do just this. The system was incomplete, since it did not indicate which regulations applied to a service; we emphasize this point in the next section. However, even this simple system demonstrated that a form of nonmonotonic reasoning could effectively be used in an industrial application.

A Brief Review of Inheritance

We briefly summarize some notation for inheritance network with exceptions (taken from (Horty, 1994)): a link between two nodes can be positive or negative. A positive (or isa) link between nodes X and Y is written $X \rightarrow Y$. A negative (or cancels) link between nodes X and Y is written $X \not\rightarrow Y$. All links are defeasible. A path is a sequence of positive links (called a *positive* path) or a sequence of positive links followed by one negative link (called a *negative* path). The notation $\pi(x, \sigma, y)$ (resp. $\pi'(x, \sigma, y)$) represents a positive (resp. negative) path from *x* to *y* through the path σ. If there are positive and negative paths between two nodes, we follow the analysis of Touretzky (1986) and Horty(1994) in choosing a path. Given a *context* — an inheritance network *T* and a set of paths $, a path is *inheritable* or *undefeated* if it is *constructive* and neither *preempted* nor *conflicted.* A path is constructive in a context if it can recursively be built out of the paths in a network; a path is *conflicted* in a context if there are paths of opposite sign in the context with the same starting and ending points; a path is *preempted* if there is a conflicting path with more direct information about the path's endpoint (i.e., a direct link from an earlier point in the path).

3.3 Why Inheritance with Exceptions isn't Enough

While much of the information in the medical insurance domain — in particular, the relations among benefits and services — is taxonomic, a large chunk of information, specifically business rules, does not seem to be taxonomic in nature. This information is central to the task of benefits inquiry, since CSRs must determine which regulations apply to a service.

The problem with representing business rules in an inheritance hierarchy can best be appreciated by examining several rules. Some rules lend themselves to representation within a semantic network. Consider the rule:

There is a co-pay of 20% for diagnostic services

To represent this rule using the standard inheritance network model, one could have a node representing the services which have a 20% co-pay, and a subtype link between the Diagnostic Services node and this node.

On the other hand, a more complex rule such as

Patients in Drug Rehabilitation programs lose all rehab benefits for a year if they are non-compliant

Drugs, and Insulin Syringes and Supplies. In this case we allow prioritisation of a particular path.

cannot be so easily represented. One could posit a node that represents the services which have the property that if patients are non-compliant with respect to that service, then they lose all benefits for a year, and then have a subtype link between the Drug Rehab Services node and this node. But such a node appears quite artificial and outside the spirit of a semantic network, where nodes are supposed to represent easily understood concepts.

The fact that much of the domain knowledge is not taxonomic in nature means that we must go outside of the standard structure of an inheritance network with exceptions. On the other hand, it is desirable to build on the inheritance network structure: first, because inheritance with exceptions already solves part of the benefits inquiry problem; second, because inheritance with exceptions is one of the few efficient nonmonotonic techniques. Furthermore, there is an obvious connection between non-taxonomic and taxonomic knowledge in this domain. In particular, business rules often apply to particular services — i.e., to nodes in the network. Building on the existing network makes this connection explicit.

The next section suggests an extension to the network structure and explores the process of benefits inquiry in this context.

3.4 The Solution: integrating taxonomic and non-taxonomic information

Definition of a FAN

We wish to introduce a knowledge structure that is capable of representing both taxonomic and non-taxonomic information. The aim is to represent the taxonomic information in a standard inheritance network with exceptions and to attach the non-taxonomic information to the network in some way. For the (instance of the) medical insurance domain, we would like to represent the fact that business rules generally apply to specific medical services and benefits by attaching business rules to nodes in the network. To do this, we introduce the concept of a formula-augmented semantic (or inheritance) network (FAN), an inheritance network in which sets of logical formulae may be attached to nodes. In the medical insurance domain, the logical formulae usually represent business rules, but they could also represent other sorts of information, including lists or tables.

Formally, a FAN is a tuple $(N, W, B, \mathcal{L}1, \mathcal{L}2, \mathcal{O})$, where

- N is a set of nodes. A node represents some set of medical services; e.g., Physical Therapy represents the set of physical therapy services. A node may represent a set of services that are covered by a particular benefit, as in the root nodes of Figure 1.
- The set of wffs W consists of well-formed formulae of a sorted first-order logic.
- The background *B* is a (possibly empty) set of wffs of first-order logic, intuitively representing the background information that is true. In the medical insurance domain, it includes all rules that are true of all medical services and benefits. It may also include patients' medical records and pay scales. In general, it consists of non-taxonomic information that is too general to attach

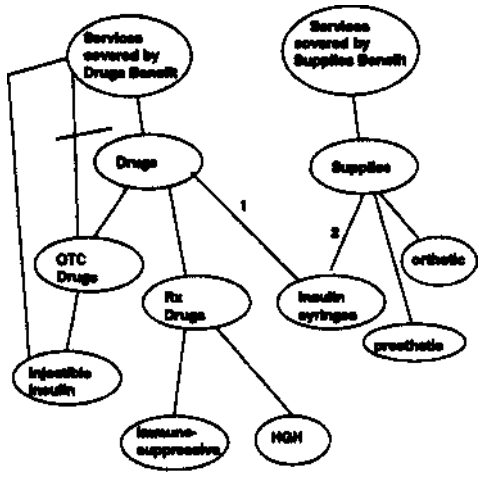

Figure 1: *A portion of the medical insurance network. Lines represent isa links; slashed lines represent cancels links. Note the presence of non-conflicting multiple path inheritance, and the ordering placed on links in the network.*

to a specific node in the network.

- £1 is the set of links on nodes, as described in sec. 3.2.
- *O,* the ordering on links, gives a preference on links. This is useful for non-conflicting multiple path inheritance, since it allows us to prefer one path over another.
- £2 is the set of links connecting nodes and sets of wffs. If N is a node and W is a set of wffs, N —► W means that the set of wffs W is attached to node N. Intuitively, this means that each wff of W is typically true at N.

Inheriting Well-formed Formulae

CSRs must determine which set of business rules applies to a medical service or benefit. This translates into determining which wffs *apply* to a node. Note that determining which wffs *apply* to a node is not the same as determining which wffs are *attached* to a node; the latter is a trivial operation. For example (Figure 4), assume that there is a cost-share rule attached to the Therapy node, specifying that the co-pay is 20%, and a rule attached to the Physical Therapy node, specifying a maximum of twenty visits a year without a doctor's written prescription. It seems clear that the cost-share rule attached to Therapy also applies to Physical Therapy, since Physical Therapy is a subtype of Therapy. That is, Physical Therapy in some sense *inherits* wffs from Therapy.

The process of inheriting wffs is considerably more complex than standard attribute inheritance. One might think that wff-inheritance is performed in the following manner: To determine which wffs apply to a node N, compute all nodes Ni such that there is an undefeated positive path from N to Ni. Then take the union *U* of all wffs attached to all such nodes Ni. This suggestion, however, leads to inconsistency, as Figure 2 shows. Since there is an undefeated path from N3 to N1, we would get

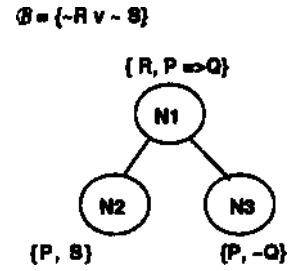

Figure 2: *Taking the union of wffs at nodes yields inconsistency.*

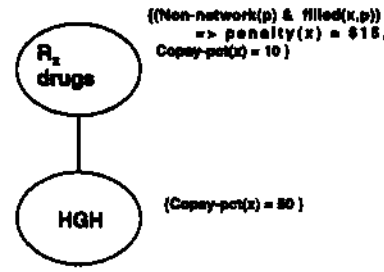

Figure 3: *The wffs at HGH are more specific than the wffs at R_x, drugs and are thus preferable.*

$\mathbf{wffs(N3) \cup wffs(N1) = \{P, \neg Q\} \cup \{R, P \Rightarrow Q\}}$, which is obviously inconsistent. Note also that the procedure will not work correctly for node N2; although wffs(N2) U wffs(Nl) is consistent, it is inconsistent with respect to the background B.

Rather, the wffs that apply to a node are a *maximally consistent subset* of U.[10] There may be many maximally consistent subsets of *U;* some of these are obviously preferable to others. For example, in Figure 3, the union of rules at HGH Drugs is inconsistent. We have the choice to construct a maximally consistent subset by throwing out the cost-share rule at Prescription Drugs or by throwing out the cost-share rules at HGH Drugs. Intuitively, we would rather keep the cost-share rule at HGH Drugs since it is more specific than the rule at Drugs. Thus, we *prefer* the maximally consistent subset

{ Non-network(p) & filled(x,p) ⇒ penalty(x) = $15, Copay-pct(x) = 50 }

to the maximally consistent subset

{ Non-network(p) & filled(x,p) ⇒ penalty(x) = $15, Copay-pct(x) = 10 }.

It is known as a *preferred maximally consistent subset*

In general, we prefer wffs from nodes that are more specific and/or on preferred paths. Thus, for example (Figure 4), when one is computing the set of wffs which apply to Cardiac Rehab, the wffs attached to Cardiac Rehab are preferable to the wffs attached to Physical

[10] A maximally consistent subset of *U* is a consistent subset S that is maximal; that is, if S' is a consistent subset of *U,* it is not the case that $\mathbf{S \subset S'}$.

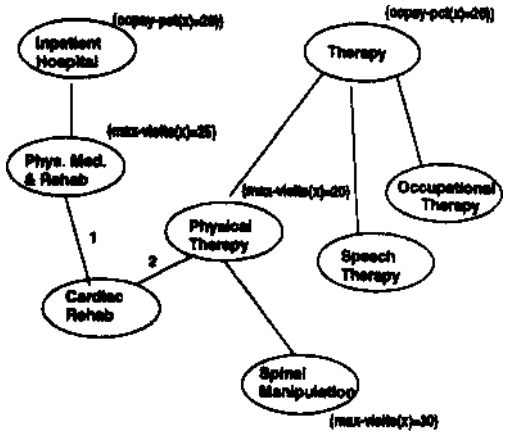

Figure 4: *Preferences based on specificity and path ordering: Spinal Manipulation is preferred to Physical Therapy and Physical Therapy to Therapy due to specificity; from Cardiac Rehab's point of view, PM&R is preferable to Physical Therapy due to path ordering.*

Therapy, which are preferable to the wffs attached to Therapy. In addition, the wffs attached to PM&R (Physical Medicine and Rehabilitation) are preferable to the wffs attached to Physical Therapy. Thus, the preferred maximally consistent set of wffs in this case is {Max-visits(x)=25, Copay-pct(x)=20%}.

Other preference criteria may also be desired; for example, one may wish to assign some wffs a higher priority than others (as in (McCarthy, 1986)), regardless of the rule's position in the network; for example, medical rules might have higher priorities than administrative rules. Likewise, one may prefer a subset of rules based on what the rules entail; this is equivalent to preferring one extension or model to another (as in (Shoham, 1988)). These criteria have not, however, been implemented in the current system.

Preferred maximally consistent sets are not necessarily unique.

The Algorithm

How do we compute a preferred maximally consistent subset at a focus node N? First consider the simple case where there are no upward forking points (no multiple inheritance from the point of view of the focus node.) It is clear what we do *not* want to do. We do not want to first take the union of all sets of wffs at the nodes on the path from N to the root, then take maximally consistent subsets of this large set, and finally choose preferred maximally consistent subsets relative to the specificity criterion. Such a method would be extremely inefficient. Instead, we want to iteratively traverse the path, and perform the computation as we go along.

Upward traversal turns out to be a better choice than downward traversal. This method for traversing the network is consistent with the specificity criterion. One begins at the focus node N, taking wffs(N) (the wffs attached to N) as the starting set. One then proceeds up the path, at each node taking a preferred maximally consistent subset of the set computed so far and the wffs attached to the current node.

This process will ensure that the specificity constraint is obeyed. To ensure that path-ordering is respected in case of forking paths, we examine all links at each forking point in the path, order them, and recursively proceed up the more preferred links before the less preferred links.

We must also ensure that we do not collect rules from nodes that are only on conflicted or preempted paths; to avoid this problem, we preprocess the FAN to remove preempted and conflicted links (we do this using an extension of the procedure in (Stein, 1992) which computes the *specificity extension* at a focus node).

The complete wff-inheritance algorithm is described in (Morgenstern, 1996a).

Computational Issues

It is clear that inheriting well-formed formulae is much more computationally intensive than inheriting attributes. Inheriting attributes is polynomial; inheriting wffs (in the propositional case) is NP-hard, since computing preferred maximally consistent subsets is NP-hard. (Inheriting general first-order wffs is clearly undecidable.)

In practice this has not proven to be a real difficulty; by computing the set of inherited wffs iteratively, we deal with relatively small sets. We have also noted a possible divide-and-conquer strategy: it may be possible to divide rules into disjoint subsets, based on rule type, so that sets can contradict one another only within their own type. (This division is to a large extent natural; for example, cost-share rules never contradict medical rules.) Finally, polynomialtechniques discovered by Grosof (1997b) may be applicable to sets of rules in the system.

3.5 The Benefits Inquiry System

The Benefits Inquiry System incorporates two tools, an inquiry tool which is used by CSRs to answer customers' questions, and an authoring tool which is used by PMs to modify products. Both tools use a graphical interface which allows the user to navigate through the network and a reasoning engine which performs both attribute and wffinheritance. An early version of the system received excellent reviews from both the CSRs and PMs who used it.

4 Generalizing Wff-inheritance

Can the techniques of wff-inheritance, which were developed for the particular problem of benefits inquiry in the medical insurance domain, be generalised to other problems in industry?

Some generalisations are obvious. Wff-inheritance would clearly be useful for benefits inquiry in other parts of the insurance industry, such as life insurance and property and casualty insurance. In these industries, it is also the case that services are best organized taxonomically, and that business rules apply to services. Wff-

inheritance is also useful for other tasks in insurance, such as adjudication, which would use a taxonomy of services very close to the structure used in benefits inquiry, and administration, which would use a taxonomic structure of products as well as services.

The nonmonotonic techniques discussed here are applicable to a wide range of other problems as well. Indeed, it can be argued that the construct of a FAN and its associated algorithms may prove useful in other domains which satisfy the following criteria:

1. there exists a large amount of taxonomic information
2. there exists a significant amount of non- taxonomic information, conceptually linked to the taxonomic information
3. the non-taxonomic information can be mapped into wffs

There are a number of potential domains:

• Legal Reasoning, especially case law: Legal cases are often organized taxonomically, and different legal rulings are associated with cases; it seems that these legal rulings can be mapped into wffs. Most automated legal reasoning has been case-based (Ashley, 1991), or analogical; adding wff-inheritance may significantly enhance the power of legal reasoning systems.

• Medical Reasoning and Treatment: Medical conditions are often organised taxonomically, and protocols are associated with these conditions. The protocols are often rigorous sequences of steps which can be mapped into wffs.

• Reasoning in Business Organizations: The organisation chart in many businesses is a perfect taxonomy, and there are many rules associated with different positions in the org chart.

These extensions are not straightforward; in particular, mapping business or legal rules into wffs is nontrivial. However, these examples indicate that the usefulness of FANs extends far beyond the domain for which they were invented.

5 Into the Future

The detailed examination of an application of nonmonotonic reasoning to industry has taught us some valuable lessons and has suggested several directions for future research.

(1) We need to constantly keep our eyes open for problems in industry that could benefit from nonmonotonic reasoning. There are many such problems; the trick is to identify them. Certainly, we should look out for problems that could benefit from FANs, as suggested above. In general, we should look for domains where exceptions are relatively common.

(2) Basic research is still crucial. We need serious research on theoretical aspects of nonmonotonic reasoning. It would be best if such research were guided by specific issues highlighted by the study of particular problems in industry. In fact, one of the unexpected dividends of intensively studying a problem in industry is that it often results in the discovery of theoretical problems that were not previously considered. For example, while I was developing the benefits inquiry system described in Section 3,1 discovered a number of issues that theories of inheritance had not previously examined. Such problems include the interaction of composition and subtyping and non-unary inheritance (Morgenstern, 1996b).

The importance of basic research cannot be overstated. Some of the most heartening news about the current state of nonmonotonic research is the recent spate of exciting results regarding the complexity of some restricted nonmonotonic theories. Examples include the result that computation of the answer set for courteous logic programs, a restricted form of prioritised defaults, is $0(n^2)$ (Grosof, 1997b). These results are being translated into commercial products. In particular, courteous logic programs are used in RAISE, a system for building intelligent agents, now commercially released (Grosof, 1997a).

(3) The proper balance between basic research and serious involvement in industry is important, but difficult to maintain. One meaty industry problem can easily give a theoretical researcher enough material for a decade; on the other hand, we need to work on many industrial problems to get a fair idea of the problems that need to be solved, and to convince industry of the relevance of nonmonotonic reasoning.

(4) We must develop tools to perform nonmonotonic reasoning. We need to develop general tools for inheritance with exceptions; we also need to develop a tool for general inheritance with wffs. Thus far, inheritance with wffs has been developed for only one application and modified for another. Such a tool will facilitate the extension of FANs to other problems in industry, as suggested above.

Finally, we must keep in mind that researchers in nonmonotonic reasoning do not always face a friendly landscape. Some things we ought to watch out for:

1. Shortsightedness. It always takes longer to solve a problem well, especially the first time. Using nonmonotonic reasoning takes a lot more time, and the advantages may not always be obvious to anyone but AI researchers. AI researchers should be prepared for the possibility of an uphill battle, both with one's management chain and with the customer. This isn't a problem unique to the field of nonmonotonic reasoning, of course.
2. Refusing to accept the importance of plausible reasoning. This comes in many guises:

(i) The 80-20 rule. This line of argument runs as follows: Even if we ignore exceptions, we'll still get things right most (around 80%) of the time, and with very little effort. Isn't it worth taking that route?

The 80-20 rule is particularly pernicious if one is willing to accept wrong answers 20% of the time (one can only hope that this rule is not invoked by the FAA), but is quite troublesome even if one is merely willing to accept admissions of ignorance (answers of "I don't know") 20% of the time. Even in the relatively benign domain of benefits inquiry, a system that can't answer questions 20% of the time is not very useful: its performance would scarcely be better than the desktop systems that are de-

signed to answer the most frequently asked questions, or CSRs without any aid of technology who can generally answer frequently asked questions right off the bat.
(ii) The back-to-if-then-else movement. This argument recognises the importance of exceptions, but insists that any branching statement is all that is needed. People who use this argument are convinced that all that nonmonotonic reasoning is trying to achieve has been present since the days of Algol 60 or earlier,
(iii) The protection-of-basic-researchers strategy. Despite constantly urging nonmonotonic researchers to do something practical, management often tries to keep a buffer between researchers and industry. The trouble with this is that if researchers can't get close enough to industry, they can't find the problems that are most suitable; if they only hear about a problem second-hand, they don't have an accurate picture of the situation, and they can't determine whether and how nonmonotonic reasoning is useful.

The best way to counteract these obstacles is to demonstrate that nonmonotonic reasoning is capable of yielding practical results. We will achieve recognition when we affect the outside world.

Acknowledgements: I am grateful to Ernie Davis, Benjamin Grosof, and Moninder Singh for helpful discussions and suggestions.

References

K. Ashley (1991). *Modeling Legal Argument: Reasoning with Cases and Hypotheses,* MIT Press, Cambridge.

W. F. Clocksin and C.S. Mellish (1987). *Programming in Prolog,* 3rd edition, Springer Verlag, Boston

E. Davis (1990). *Representations of Commonsense Knowledge,* Morgan Kaufmann, San Mateo.

W.H. Dowling and J.H. Gailier(1984). Linear Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, *Journal of Logic Programming 8,* 267-284.

H. Geffner(1990). *Default Reasoning: Causal And Conditional Theories,* MIT Press, Cambridge

M. Ginsberg, ed (1987). *Readings in Nonmonotonic Reasoning,* Morgan Kaufmann, San Mateo.

G. Gottlob (1996). Complexity and Expressive Power of KR Formalisms, *Principles of Knowledge Representation and Reasoning: Proceedings of the 5th Intn'l Conference (KR-96),* Morgan Kaufmann, San Francisco, 647-649.

Grosof (1997a). Building Commercial Agents: an IBM Research Perspective, *Proc, 2nd Intn'l Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM97).* Available as IBM Research Report RC20835 and at http://www.research.ibm.com/people/g/grosof.

B. Grosof (1997b). Proritized Conflict Handling for Logic Programs, IBM Research Report RC20836. Also at http://www.research.ibm.com/people/g/gro8of.

S. Hanks and D. McDermott (1986). Default Reasoning, Nonmonotonic Logic, and the Frame Problem, *Proceedings, AAAI-1986,* 328-333.

J. Horty (1994). Some Direct Theories of Nonmonotonic Inheritance in D. Gabbay, C. Hogger, and J. Robinson, eds: *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 8: Nonmonotonic Reasoning and Uncertain Reasoning,* Oxford University Press, Oxford, 111-187.

J. Horty, R. Thomason and D. Touretzky (1990). A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks, *Artificial Intelligence 42,* 311-349.

W. Kneale and M. Kneale (1962). *The Development of Logic,* Clarendon Press, Oxford.

S. Kraus, D. Lehman, and M. Magidor (1990). Nonmonotonic Reasoning, Preferential Models, and Cumulative Logics, *Artificial Intelligence 44,* 167-207.

E. Mays, R. Dionne, and R. Weida(1991). K-REP System Overview, *SIGART Bulletin,* 2:3.

J. McCarthy(1980). Circumscription - A Form of Non-Monotonic Reasoning, *Artificial Intelligence 18,* 27-39.

J. McCarthy (1986). Applications of Circumscription to Formalizing Common-sense Knowledge, *Artificial Intelligence 28,* 86-116.

D. McDermott and J. Doyle (1980). Nonmonotonic Logic I. *Artificial Intelligence 18,* 41-72.

L. Morgenstern (1996a). Inheriting Well-formed Formulae in a Formula-Augmented Semantic Network, *Principles of Knowledge Representation and Reasoning: Proceedings of the 5th Intn'l Conference (KR-96),* 268-279.

L. Morgenstern (1996b). New Problems for Inheritance Theories, *Working Papers, Third Symposium on Formal Theories of Commonsense Reasoning,* Stanford, January 1996. Available at http://www-formal.stanford.edu/leora.

L. Morgenstern (1996c). The Problem with Solutions to the Frame Problem, in K. Ford and Z. Pylyshyn, eds: *The Robot's Dilemma Revisited,* Ablex, 1996, 99-133.

L. Morgenstern and M. Singh (1997). An Expert System Using Nonmonotonic Techniques for Benefits Inquiry in the Insurance Industry, *Proceedings, IJCAI-97.*

R. Reiter (1980). A Logic for Default Reasoning, *Artificial Intelligence 18,* 81-132.

J. Schmolze and T. Lipkis (1983). Classification in the KL-ONE Representation System, *Proc, IJCAI-1988.*

B. Selman and H. Levesque (1993). The Complexity of Path-Based Defeasible Inheritance, *Artificial Intelligence 62:2,* 303-340.

Y. Shoham (1988). *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence,* MIT Press, Cambridge.

L. Stein (1992). Resolving Ambiguity in Nonmonotonic Inheritance Hierarchies, *Artificial Intelligence 55,* 259-310.

D. Touretsky (1986). *The Mathematics of Inheritance Systems,* Morgan Kaufmann, Los Altos.

L. Zadeh (1992). Fussy Sets, in D. Dubois, H. Prade, and R. Yager, eds. *Readings in Fuzzy Sets and Intelligent Systems,* Morgan Kaufmann, San Mateo.