A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks

John F. Horty[1,3]
Richmond H. Thomason[2]
David S. Touretzky[3]

October 1987
CMU-CS-87-175

[1]Philosophy Department
University of Maryland
College Park, MD 20742

[2]Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260

[3]Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based on work supported by the National Science Foundation under Grants No. IST-8516313 and IRI-8700705.

ABSTRACT

This paper describes a new approach to inheritance reasoning in semantic networks allowing for multiple inheritance with exceptions. The approach leads to an analysis of defeasible inheritance which is both well-defined and intuitively attractive: it yields unambiguous results applied to any acyclic semantic network, and the results conform to our intuitions in the cases in which the intuitions themselves are firm and unambiguous. Since the definition provided here is based on an alternative, skeptical view of inheritance reasoning, however, it does not always agree with previous definitions when it is applied to nets about which our intuitions are unsettled, or in which different reasoning strategies could naturally be expected to yield distinct results. After exploring certain features of the definition presented here, we describe also a hybrid (parallel-serial) algorithm that implements the definition in a parallel marker-passing architecture.

CONTENTS

1 Introduction

This paper describes a new approach to inheritance reasoning in semantic networks allowing for multiple inheritance with exceptions. Like the previous approaches of Touretzky [16] and Etherington [4], but unlike many other approaches, such as those of Roberts and Goldstein [13] or Fahlman [5], the approach presented here leads to an analysis of defeasible inheritance which is both well-defined and intuitively attractive: it yields unambiguous results applied to any acyclic semantic network, and the results conform to our intuitions in those cases in which our intuitions themselves are firm and unambiguous. Since the definition provided here is based on an alternative, skeptical view of inheritance reasoning, however, it does not always agree with these previous definitions when it is applied to nets about which our intuitions are unsettled, or in which different reasoning strategies could naturally be expected to yield distinct results.

The paper is organized as follows. In Section 2, after setting out our notation and basic terminology, we sketch a view of the general nature of inheritance reasoning, drawing upon a loose analogy between inheritance reasoning and ordinary deductive reasoning. In Section 3, we isolate the principles underlying our particular approach to inheritance. These principles are then organized into a rigorous definition in Section 4; the resulting definition is examined in more detail in Section 5. An actual inheritance reasoner based on the definition presented here has been implemented in Common Lisp; the program, along with some sample inputs and runs, is contained in an appendix. Of more theoretical interest, we describe in Section 6 a hybrid (parallel-serial) algorithm that implements the inheritance definition presented here in a parallel marker-passing architecture.

We do not attempt in this paper to provide any systematic comparison of our approach to nonmonotonic inheritance either with those of Touretzky [16] and Etherington [4], or with other similar approaches to nonmonotonic reasoning; this project of comparison and evaluation is begun in [17], where we set out a partial design space for the classification of inheritance systems and investigate the consequences of some different design choices. However, one aspect of our account deserves to be mentioned. While the previous theories of Touretzky and Etherington (since they are based on a fixed point construction) tend to associate a number of of different extensions with a single network, the skeptical theory described in this paper always leads to a unique extension. Because it allows us to avoid the complexities of dealing with multiple extensions, this skeptical approach may prove to be more practical in some applications.

2 Basic concepts

2.1 Notation

Letters from the beginning of the alphabet (a, b, c) will represent *objects*, and letters from the middle of the alphabet (p, q, r) will represent *kinds* of objects. Letters from the end of the alphabet (u, v, w, x, y, z) will range over both objects and kinds.

An *assertion* will have the form $x \rightarrow y$ or $x \not\rightarrow y$, where y is a kind. If x is an object, such an assertion should be interpreted as an ordinary atomic statement: $a \rightarrow p$ and $b \not\rightarrow p$, for instance, are analogous to Pa and $\neg Pb$ in logic; they might represent statements like 'Tweety is a bird' and 'Jumbo isn't a bird'. If x is a kind, these assertions should be interpreted as *generic* statements: $p \rightarrow q$ and $r \not\rightarrow q$, for example, might represent the statements 'Birds fly' and 'Mammals don't fly'. There is nothing in ordinary logic very close in meaning to generic statements like these, since they can be true even in the presence of exceptions. In particular, 'Birds fly' can't be interpreted to mean $\forall x[Px \supset Qx]$, and 'Mammals don't fly' doesn't mean anything like $\forall x[Rx \supset \neg Qx]$.[1] We describe a pair of assertions having the form $x \rightarrow y$ and $x \not\rightarrow y$ as *conflicting assertions*. Note that the conflicting assertions include not only logically contradictory pairs, like 'Tweety is a bird' and 'Tweety isn't a bird', but also the kind of conflicts exhibited by pairs of generic statements such as 'Birds fly' and 'Birds don't fly'.[2]

Capital Greek letters will represent *networks*, where a network consists of a set I of individuals and a set K of kinds, together with a set of positive links and a set of negative links, both finite subsets of $(I \times K) \cup (K \times K)$. We identify the positive and negative links in a network with our positive and negative assertions.

Lower case Greek letters will range over sequences of links, among which we single out for special consideration the *paths*, defined inductively as follows: each assertion is a path; and if $\sigma \rightarrow p$ is a path, then both $\sigma \rightarrow p \rightarrow q$ and $\sigma \rightarrow p \not\rightarrow q$ are paths. As this notation indicates, paths are special kinds of link sequences—joined, in the sense that the end node of any link in a path is identical with the initial node of the next link. It follows from their definition that paths are subject also to two further constraints. First, a negative link can

[1] For detailed argumentation on this point, with supporting linguistic evidence, see Carlson [3].

[2] Intuitively, these pairs are inconsistent. But whether the second is the negation of the first, and whether they are logically inconsistent, are issues that would have to be settled by a logic of the generic plural. Unfortunately, there is as yet no such logic.

occur in a path, if at all, only at the very end: $a \rightarrow p \not\rightarrow q$ is a path, but $a \not\rightarrow p \rightarrow q$ isn't. Second, an individual can occur only as the initial node of a path: $p \rightarrow a \not\rightarrow q$ isn't a path.

Paths will be said to *enable* assertions, or statements, much in the way that proofs enable their conclusions: a path of the form $x \rightarrow \sigma \rightarrow y$ is said to enable the assertion $x \rightarrow y$, and likewise, a path of the form $x \rightarrow \sigma \not\rightarrow y$ is said to enable the assertion $x \not\rightarrow y$. As this suggests, it is often natural to understand a path—like a proof—as representing a particular chain of reasoning behind the assertion it enables. The path $a \rightarrow p \rightarrow q$, for example, enables the assertion 'Tweety flies', while representing an argument like "Tweety flies because he is a bird and birds fly." We describe describe a pair of paths as *conflicting paths* if they enable conflicting assertions.

2.2 Inheritance

Since we identify the links in a net with assertions, a net can be viewed as a set of hypotheses, or axioms. Let us say that an assertion A is *supported* by a net Γ if we can reasonably conclude that A is true whenever all the links in Γ are true—if the information contained in Γ would naturally lead to the conclusion that A. We want to know what we can conclude from a given net; so our object is to define the general conditions under which a net Γ supports an assertion A.

In the context of ordinary deductive logic, we often find ourselves in a similar situation, when we want to know what statements are *deducible* from a given set of hypotheses. In that context, it is a common practice to approach the question in a roundabout way. Instead of defining the relation of deducibility directly, one first characterizes the deductions—sequences of statements representing certain kinds of arguments, or chains of reasoning—and then defines a statement as deducible from a set of hypotheses if those hypotheses permit a deduction of that statement.

Of course, the process of drawing conclusions from a set of hypotheses through inheritance reasoning is quite different from the process of drawing conclusions through deduction. Inheritance reasoning doesn't depend on the interplay of connectives, for example, since there aren't really any connectives, to speak of, in our semantic nets; and even some of the connective-free structural rules governing classical deducibility (such as Weakening) fail to hold for nonmonotonic inheritance. Still, we find it helpful in the case of inheritance to follow a similar kind of roundabout strategy in describing the consequences of a set of hypotheses. Instead of trying to specify directly the statements supported by

a given net, we first characterize the arguments or chains of reasoning—represented, now, by paths—that are *permitted* by a net. As in the case of ordinary deducibility, this relation between sets of hypotheses and the chains of reasoning they permit is really the central idea; and it will be the primary focus of our attention. Once we have identified the paths that a net permits, it is natural to define the statements supported by a net by stipulating that a net *supports* a statement just in case it *permits* a path *enabling* that statement.

We refer to the entire set of statements a net supports as the *theory* of the net, and to the entire set of paths it supports as its *extension*.

3 Motivation

In this section we examine several simple examples of nets and the paths they should permit, in order to illustrate the principles underlying our general characterization of the permission relation, which is then presented in Section 4.

3.1 Forward chaining

Consider, first, the simplest kind of case imaginable, a linear net Γ_1 (Figure 1). Just to fix an interpretation, let a = Tweety, p = Canaries, q = Birds, and r = Flying Things. Γ_1 explicitly contains the information, then, that Tweety is a canary, that canaries are birds, and that birds fly. Now given just this information, we would certainly want to allow a chain of reasoning along the lines of "Since Tweety is a canary, a kind of bird, and birds fly, Tweety flies"—so we want the net Γ_1 to permit the compound path $a \rightarrow p \rightarrow q \rightarrow r$, representing this argument. In just the same way, we want the net Γ_2 (Figure 2), with b = Jumbo, s = Royal Elephants, t = Elephants, and u = Flying Things, to permit the path $b \rightarrow s \rightarrow t \not\rightarrow u$, which represents an argument something like "Jumbo is a royal elephant, a kind of elephant, and elephants don't fly; so Jumbo doesn't fly."

These examples illustrate some of the compound reasoning paths that can be constructed by assembling the direct links contained in a net, but they don't yet tell us, when we think of the construction as proceeding inductively, *how* these paths are to be assembled. There are, of course, two natural options for assembling compound paths from direct links: roughly, top-down and bottom-up. Most treatments of inheritance reasoning—including those of Roberts and Goldstein [13], Fahlman [5], and Touretzky [16]—presume the top-

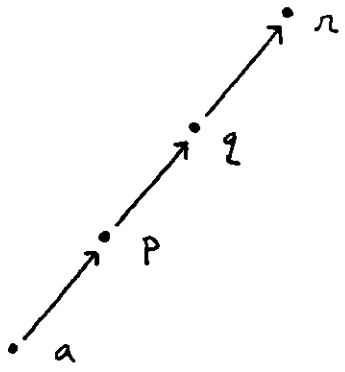

Figure 1: Γ_1

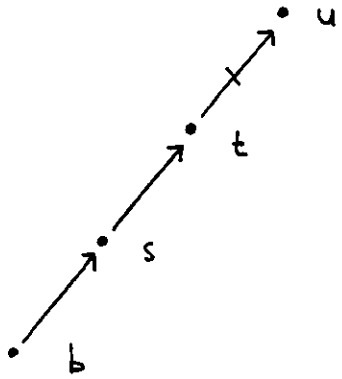

Figure 2: Γ_2

down approach. They are guided, more or less explicitly, by a picture of inheritance according to which properties are imagined to flow downward through the semantic net, from more general to more specific kinds and then finally to individuals, unless the flow is interrupted, somehow, by an exception. Formally, this "property flow" picture leads to the construction of compound permitted paths through the process of *backward chaining*, according to which, at the inductive step, a compound permitted path of the form $x \rightarrow y \rightarrow \sigma$ is assembled by adding the direct link $x \rightarrow y$ to its permitted end segment $y \rightarrow \sigma$.

The present treatment, on the other hand, is intended to capture a kind of bottom-up approach to inheritance reasoning. This approach seems especially natural when one wants to push the analogy, as we do, between paths and arguments—since arguments, at least as they are usually represented (say, by proof sequences), tend to move from the beginning forward. Formally, the bottom-up approach leads to the construction of compound paths through the process of *forward chaining*: at the inductive step, the compound permitted path $\sigma \rightarrow x \rightarrow y$ is assembled by adding the direct link $x \rightarrow y$ to its permitted initial segment $\sigma \rightarrow x$; and likewise, the compound permitted path $\sigma \rightarrow x \not\rightarrow y$ is assembled by adding the direct link $x \not\rightarrow y$ to the permitted initial segment $\sigma \rightarrow x$. This adherence to forward chaining is one of the central principles guiding our approach. Not only does it embody a different metaphor for inheritance reasoning ("argument construction" instead of "property flow"), but it leads also to different technical results, as illustrated by our

discussion of the net Γ_{15} in Section 5.5, below.[3]

3.2 Restricted skepticism

In our approach, then, compound permitted paths are constructed through forward chaining, but of course, not every path constructible through forward chaining from the materials in a given net should be permitted by that net. Conflicts can interfere, as in the net Γ_3 (Figure 3). This net has come to be known as the Nixon Diamond, because of the interpretation, due to Reiter, under which a = Nixon, q = Quakers, r = Republicans, and p = Pacifists. What Γ_3 tells us explicitly, under this interpretation, is that Nixon is both a Quaker and a Republican, that Quakers are pacifists, and that Republicans are not pacifists. Unrestricted forward chaining would allow us to construct from this information both the paths $a \rightarrow q \rightarrow p$ and $a \rightarrow r \not\rightarrow p$. But since these two paths conflict, enabling the conflicting statements $a \rightarrow p$ and $a \not\rightarrow p$, we don't want Γ_3 to permit both these paths at once. Given just the information contained in Γ_3, we wouldn't want to conclude both that Nixon is a pacifist and that he isn't.

What you say about inheritance depends crucially on your treatment of nets like the Nixon Diamond, which contain compound conflicting paths. One option is to suppose, although you can't permit both of two such paths, that it is reasonable to permit one or the other. In the case of the Nixon Diamond, for example, this strategy would lead us to the conclusion that *either* the path $a \rightarrow q \rightarrow p$ *or* the path $a \rightarrow r \not\rightarrow p$ should be permitted. What lies behind this strategy is a kind of *credulity* or *belief-hunger*—the idea that it's best to draw as many conclusions as possible from a given net, even at the cost of making arbitrary choices among conflicting arguments. As developed by Touretzky [16], this strategy involves associating with each net containing compound conflicting paths a number of consistent extensions, which he refers to as "grounded expansions," reminiscent of the "extensions" of Reiter [12] or the "fixed points" of McDermott and Doyle [10]. For this reason, because they can consistently be associated with a number of different

[3]Of course, the very description of the kind of network-based reasoning we study here as "inheritance" reasoning itself seems to suggest the top-down or "property flow" picture—according to which individuals are supposed to "inherit" properties from their superiors in a network roughly as one might inherit, say, the family jewels from Aunt Martha. Once one adopts the bottom-up approach, the terminology of "inheritance" is no longer so appropriate; but the terminology by now has become fixed, and it would introduce more confusion than it would eliminate if we tried to characterize this kind of reasoning process in a phrase more nearly neutral between the top-down and the bottom-up views.

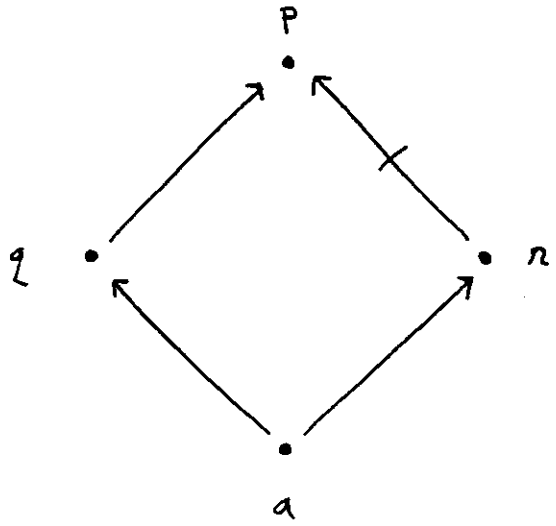

Figure 3: Γ_3

extensions, Touretzky describes nets like these as "ambiguous."

We take a different point of view. Rather than supposing that an inheritance reasoner should try to conclude as much as possible from a given net, we adopt a broadly *skeptical* attitude, according to which conflicting arguments tend to neutralize each other. Our basic idea, which will have to be explained in more detail, is that *a compound path is neutralized by any conflicting path which is not itself preempted.* Even without further explanation, however, some of the consequences of this basic skeptical intuition should be clear. Given just the information in the Nixon Diamond, for example, an inheritance reasoner guided by our skeptical reasoning strategy, won't be able to conclude either that Nixon is a pacifist or that he isn't. It won't conclude that he is a pacifist, since the information contained in the net provides the materials for constructing an argument to the contrary; it won't conclude that he isn't a pacifist, since the net also provides the materials for constructing an argument that he is.

Although our approach is based, generally, on the skeptical idea that such paths tend to neutralize each other, the special brand of skepticism we adopt here is restricted in two ways. First, we suppose that only *compound* paths can be neutralized at all; and second, that paths can be neutralized only by conflicting paths which are not themselves *preempted.* Both of these restrictions are important; we examine them in turn.

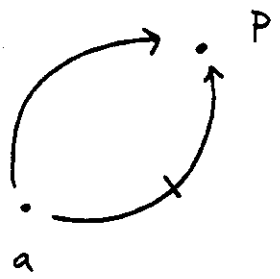

Figure 4: Γ_4

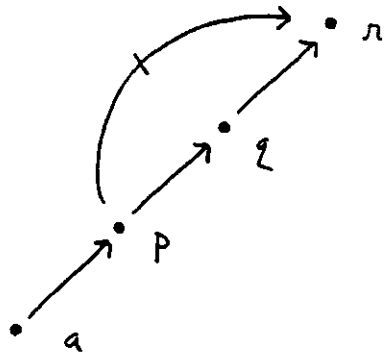

Figure 5: Γ_5

3.2.1 Compound vs. direct conflicts

As an example of a net containing conflicting direct paths, consider Γ_4 (Figure 4). (Again, take a = Nixon and p = Pacifists.) According to the definition we provide, Γ_4 will permit *both* the conflicting paths $a \rightarrow p$ and $a \not\rightarrow p$: our reasoner will conclude from Γ_4 both that Nixon is a pacifist and that he isn't. This may seem odd, especially in light of our cautious, skeptical approach to Γ_3. It may appear, from a certain point of view, that Γ_4 presents us with nothing but a limiting case of the phenomenon found in Γ_3—so that consistency of principle should lead us to conclude, if Γ_3 doesn't permit either the path $a \rightarrow q \rightarrow p$ or the path $a \rightarrow r \not\rightarrow p$, that Γ_4, likewise, shouldn't permit either of the paths $a \rightarrow p$ or $a \not\rightarrow p$. But it is also possible to isolate a point of view from which our different treatment of the conflicting paths in Γ_3 and Γ_4 seems just right.

Remember, we are talking about the design of an inheritance reasoner, a mechanism for drawing conclusions from a certain kind of database—a set of statements that can be represented as the set of links in a net. Now when we think of the net Γ_3 as a database, it is, of course, consistent: in fact, under the Nixon interpretation, all of the statements contained in Γ_3 are true. Obviously, no one would want a reasoning mechanism to draw inconsistent conclusions from consistent information; so it follows at once that Γ_3 *can't* permit both the paths $a \rightarrow q \rightarrow p$ and $a \rightarrow r \not\rightarrow p$, since these two paths enable the conflicting statements that Nixon is a pacifist ($a \rightarrow p$) and that he isn't ($a \not\rightarrow p$). On the other hand, when we look at Γ_4 as a database, it already *contains* both of these statements;

so in this case, we are faced with the problem of drawing the appropriate conclusions from information that is already inconsistent.

This is a notoriously difficult problem, but we find that it is both possible and useful to adopt in the context of inheritance reasoning a proposal that was originally formulated by Belnap, in [1] and [2], as a guide for *deductive* reasoning in the presence of inconsistency. As a general principle, we propose that a reasoner ought to be able to conclude from a set of statements every statement actually contained in that set, at least—even if the set is inconsistent. It follows, of course, that if our inheritance reasoner were actually provided with the information contained in Γ_4—that Nixon both is and isn't a pacifist—it ought to be able to *conclude* from this information both that Nixon is a pacifist and that he isn't. Thinking of deductive reasoning, Belnap argues that the presence of inconsistent information shouldn't enable a mechanical reasoner to derive arbitrary conclusions, as it would in the case of a theorem prover using classical logic. We have shown in [15], however, that this much of the motivation behind relevance logic is already built into inheritance reasoning, even in the simple case of monotonic inheritance; and the arguments we provided there carry over into the nonmonotonic case. Thus, the reasoner we describe in this paper will conclude from Γ_4, as it should, both that Nixon is a pacifist and that he isn't; but it won't then go on to draw irrelevant conclusions from this conflict: it won't conclude, for instance, that Nixon is a Democrat.

3.2.2 Preemption

The second restriction on our broadly skeptical outlook is the idea that even compound arguments are neutralized only by those conflicting arguments that are not themselves preempted. This idea—that certain compound arguments can be, as we say, *preempted* by others—really lies at the heart of our approach, allowing us to transform a simple and dogmatic skepticism into something much more interesting.

Again, we begin with an example, the net Γ_5 (Figure 5). This net results from adding the link $p \not\rightarrow r$ to Γ_1, and the interpretations of these two nets will overlap as well. Just as before, we take a = Tweety, q = Birds, and r = Flying Things; but now let's shift the earlier interpretation so that p = Penguins, giving some plausibility to the new link $p \not\rightarrow r$. If things are like this, what should we conclude about Tweety: does he fly or not? Well, there are two paths to consider: $a \rightarrow p \rightarrow q \rightarrow r$, which enables the conclusion that Tweety flies, and $a \rightarrow p \not\rightarrow r$, which enables the opposite conclusion. Since both of these paths are

compound, and they enable conflicting conclusions, simple skepticism would bar us from reaching any conclusion at all. But evidently, in this case, we *should* reach a conclusion: we should conclude, in fact, that Tweety doesn't fly—since he is a penguin, and penguins don't fly. The reason we are able to conclude here that Tweety doesn't fly—even though he is a bird, and birds fly—is that penguins happen to be a specific kind of bird, so that, in case of conflicts, the information we have about Tweety in virtue of his being a penguin should override whatever we would otherwise suppose to be true of him simply because he is a bird.

This illustrates the central intuition behind preemption: *that arguments based on more specific information should be allowed to override arguments based on less specific information.* As we define it, a path will be preempted in a net, roughly, when the net provides the materials for constructing a conflicting argument based on more specific information. Looking again at the net Γ_5, we see that it both permits the path $a \rightarrow p \rightarrow q$ (telling us that Tweety is a bird) and contains the link $q \rightarrow r$ (telling us directly that birds fly). We want to say, however, that the path $a \rightarrow p \rightarrow q \rightarrow r$ (telling us that Tweety flies, because he is a bird and birds fly) is preempted in Γ_5, since the net also contains the link $p \not\rightarrow q$ (telling us directly that penguins don't fly), and conclusions deriving from the node p (penguins) are based on more specific information about a (Tweety) than conclusions deriving from the node q (birds). In terms of the topology of Γ_5, it's natural to suppose that the reason p can be said to provide more specific information about a than q does is simply that the net permits the path $a \rightarrow p \rightarrow q$ (telling us both that Tweety is a penguin and that penguins are a specific kind of bird). So, restating in a way that incorporates this analysis of specificity, we can say that the path $a \rightarrow p \rightarrow q \rightarrow r$ is preempted in Γ_5 just because there is a node p such that Γ_5 both contains the direct link $p \not\rightarrow r$ and permits the path $a \rightarrow p \rightarrow q$.

This idea of preemption can easily be generalized to apply to arbitrary nets and paths. We will say that a path of the form $x \rightarrow r \rightarrow v \rightarrow y$ is *preempted* in a net Γ just in case there is a node z such that $z \not\rightarrow y \in \Gamma$, and either $z = x$ or Γ permits a path of the form $x \rightarrow r_1 \rightarrow z \rightarrow r_2 \rightarrow v$. With exact symmetry, we say that a path of the form $x \rightarrow r \rightarrow v \not\rightarrow y$ is preempted in a net Γ just in case there is a node z such that $z \rightarrow y \in \Gamma$ and either $z = x$ or Γ permits a path of the form $x \rightarrow r_1 \rightarrow z \rightarrow r_2 \rightarrow v$.

4 Defining inheritance

Let's use the symbol '$\models$' to stand for the permission relation, so that '$\Gamma \models \sigma$' means that the net Γ permits the path σ. We have now considered the central principles underlying our approach to this idea—*forward chaining*, along with a certain kind of *restricted skepticism*. It remains only to organize these principles into a rigorous definition.

4.1 Degree

Our adoption of forward chaining suggests that a bottom-up, inductive definition should be possible. In order to frame such a definition, however, we need to be able to associate with each path σ some measure of its "complexity" in a given net Γ, in such a way that it can be decided whether $\Gamma \models \sigma$ once it is known whether $\Gamma \models \sigma'$ for each path σ' less complex in Γ than σ itself.

The natural thing to think is that we might be able to identify the complexity of a path, in this sense, with its length—but this won't work, since we will often need to know about longer paths before we can decide whether shorter paths are permitted. As an example, consider the net Γ_6 (Figure 6). Here, it follows at once from our motivating principles that the path $x \rightarrow p \rightarrow y$, which has length two, shouldn't be permitted, since it is compound and it conflicts with the unpreempted path $x \stackrel{\cdot}{\rightarrow} q \rightarrow r \not\rightarrow y$, of length three. A more complicated situation arises in the case of Γ_7 (Figure 7). Here, the path $x \rightarrow p \rightarrow y$ should be permitted: the potentially conflicting path $x \rightarrow q \rightarrow r \not\rightarrow y$ no longer interferes, since it is itself neutralized by the path $x \rightarrow s \rightarrow t \rightarrow u \not\rightarrow r$, which is longer still. Evidently, as this net suggests, before we can decide whether a particular path is to be permitted, we need to know about all the conflicting paths that might neutralize it, as well as all the paths that might neutralize those conflicting paths, all the other paths that might neutralize those, and so on.

There are several ways to order the paths in a given net so that an inductive definition, as it steps through the ordering, will wind up considering all the paths relevant to a particular path before it considers that path itself. We adopt here what seems to be the simplest such ordering; later, in Section 6.2.1, we describe a slightly more complicated ordering, with other compensating virtues. To get at our simple ordering, we first introduce an auxiliary idea. As we recall from Section 2.1, a path is a joined sequence of links containing a negative link, if at all, only at the very end. Let's say, now, that a *generalized path* is a

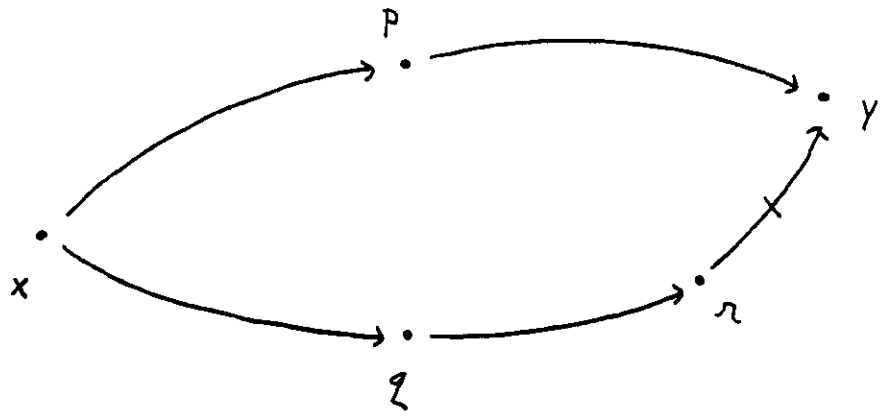

Figure 6: Γ_6

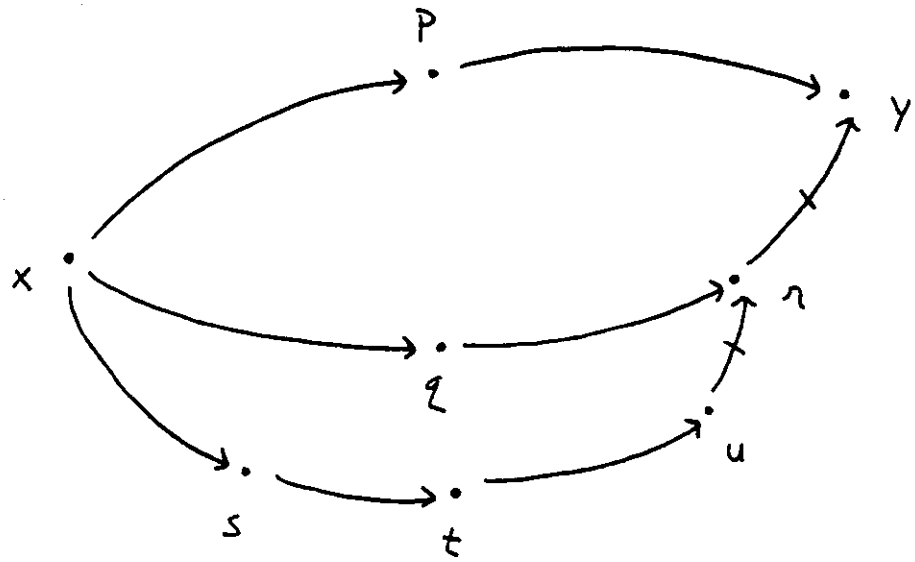

Figure 7: Γ_7

sequence of links joined like an ordinary path, except that it can contain negative links anywhere, and perhaps more than one. Formally, we can catch this idea by specifying that each assertion is a generalized path, and that, if σ is a generalized path, then both $\sigma \rightarrow y$ and $\sigma \not\rightarrow y$ are generalized paths. (It follows, of course, that the generalized paths include the ordinary paths.) Using this auxiliary concept of a generalized path, we now define the *degree of a path σ in a net Γ*—written, $\deg_\Gamma(\sigma)$—as the length of the longest generalized path in Γ from the initial node of σ to its end node. Unlike length, degree is not a property of paths alone, but of paths in a net. For example, we have

$$\deg_{\Gamma_6}(x \rightarrow p \rightarrow y) = 3,$$

$$\deg_{\Gamma_7}(x \rightarrow p \rightarrow y) = 5,$$

since in Γ_6, the longest generalized path from x to y (which happens to be an ordinary path) has length three, while Γ_7 contains a (true) generalized path from x to y of length five.

As it turns out, this idea of degree provides an acceptable notion of path "complexity" for an inductive definition of $\models$, the permission relation between nets and paths: it can be decided whether $\Gamma \models \sigma$ entirely on the basis of information regarding paths whose degree in Γ is less than that of σ, along with information about the direct links contained in Γ itself. On the other hand, in order to assure that $\deg_\Gamma(\sigma)$ should always be well-defined, we need to restrict our attention to nets which are *acyclic*, in the sense that they contain no generalized paths whose initial nodes are identical with their end nodes. (This is a common restriction; much of the analysis in Touretzky [16], for instance, also applies only to acyclic nets.)

4.2 The definition

Given this idea of degree, then, and restricting ourselves to acyclic nets, we can now present our definition of the permission relation. Although the definition is inductive at heart, it has the overall structure of a definition by cases: it deals separately with compound paths and with direct links (non-compound paths). Only in the case of compound paths is there any need to resort to induction; direct links can be handled all at once, as follows.

Case I: σ is a direct link. Then $\Gamma \models \sigma$ iff $\sigma \in \Gamma$.

It is important to note that even if σ is a direct link, it could easily turn out that $\deg_\Gamma(\sigma) > 1$, since Γ might contain a compound generalized path from the initial node of σ to its end node. On the other hand, if $\deg_\Gamma(\sigma) = 1$, then the path σ has to be a direct link. Thus, in addition to taking care of all the direct links at once, whatever their degree, Case I serves also as the basis clause for the induction on degree which extends the permission relation from direct links to compound paths. The inductive clause is as follows.

Case II: σ is a compound path with, say, $\deg_\Gamma(\sigma) = n$. As an inductive hypothesis, we can suppose it is settled whether $\Gamma \models \sigma'$ whenever $\deg_\Gamma(\sigma') < n$. There are then two subcases to consider, depending on the form of σ.

1. σ is a positive path, of the form $x \to \sigma_1 \to u \to y$. Then $\Gamma \models \sigma$ iff
   (a) $\Gamma \models x \to \sigma_1 \to u$,
   (b) $u \to y \in \Gamma$,
   (c) $x \not\to y \notin \Gamma$,
   (d) For all v such that $\Gamma \models x \to \tau \to v$ with $v \not\to y \in \Gamma$, there exists z such that $z \to y \in \Gamma$ and either $z = x$ or $\Gamma \models x \to \tau_1 \to z \to \tau_2 \to v$.
2. σ is a negative path, of the form $x \to \sigma_1 \to u \not\to y$. Then $\Gamma \models \sigma$ iff
   (a) $\Gamma \models x \to \sigma_1 \to u$,
   (b) $u \not\to y \in \Gamma$,
   (c) $x \to y \notin \Gamma$,
   (d) For all v such that $\Gamma \models x \to \tau \to v$ with $v \to y \in \Gamma$, there exists z such that $z \not\to y \in \Gamma$ and either $z = x$ or $\Gamma \models x \to \tau_1 \to z \to \tau_2 \to v$.

It should be clear that this definition of the permission relation accurately represents the general approach to inheritance reasoning described in Section 3. Case I tells us that any statement actually contained in a net should be permitted by that net. The two subcases of Case II, dealing respectively with positive and negative compound paths, are perfectly symmetric. In each subcase, the clauses (a) and (b) capture the idea of forward chaining: compound paths are permitted by a net only if they can be constructed by adding direct links from the net to initial permitted segments of those paths. The clauses (c) and (d) take care of conflicts. What (d) says is that, even if a compound path is constructible through forward chaining, it can be permitted only if each potentially conflicting compound path is preempted. Of course, only compound conflicting paths can actually be preempted, since preemption involves the intermediate nodes of a path, and

direct links have no intermediate nodes; but if, for skeptical reasons, we don't want a path to be permitted which conflicts with an unpreempted compound path, we certainly don't want to permit a path that conflicts with a direct link. This is the force of the clause (c).

Both the clauses (a) and (d) in the inductive step refer to other paths of a certain form permitted by the net; but this is no problem, because at any step in the induction, paths of this form will always have a degree less than that of the path being considered.

5 Discussion

It is easy to see that the definition of the permission relation presented in Section 4 yields the advertised results applied to the nets Γ_1 through Γ_5 from Section 3. In this section we explore certain properties of the definition. We show that the treatment of inheritance embodied in this definition is nonmonotonic, that it is sound, and that it has the property of atomic stability. We show that the skeptical theory of a network is not identical with the intersection of its credulous theories, and that it allows for decoupling of conclusions.

5.1 Nonmonotonicity

The analysis presented here is put forth as an analysis of *nonmonotonic* inheritance: paths permitted by a net may no longer be permitted once that net is supplemented with additional links; statements supported by a net may no longer be supported once that net is supplemented with additional links.

The simplest kind of counterexample to monotonicity is illustrated by the nets Γ_8 and Γ_9 (Figures 8 and 9). Here, Γ_8 permits the path $a \rightarrow q \rightarrow p$, and so supports the statement $a \rightarrow p$. On the other hand, Γ_9 neither permits the path nor supports the statement; but of course $\Gamma_8 \subseteq \Gamma_9$. From the standpoint of our general skeptical motivation, this is as it should be. Imagine that the nodes in these two nets are interpreted as in the Nixon Diamond, with a = Nixon, q = Quakers, and p = Pacifists. Then Γ_8 gives us the materials for constructing an argument, represented by the path $a \rightarrow q \rightarrow p$, for the conclusion that Nixon is a pacifist; but in the net Γ_9, both this argument and its conclusion are neutralized by the direct statement that Nixon is not a pacifist.

A different kind of counterexample to monotonicity—relying on preemption rather than a conflict with direct links—is provided by the nets Γ_1 and Γ_5 from Section 3. Again, Γ_1

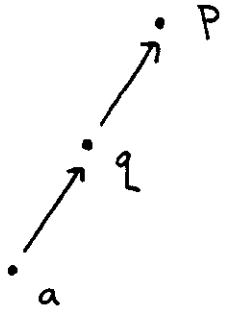

Figure 8: Γ_8

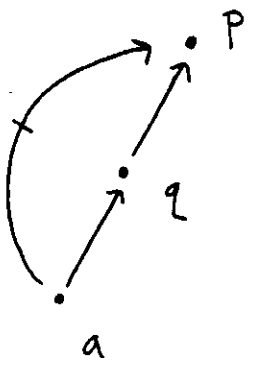

Figure 9: Γ_9

is a subset of Γ_5. However, Γ_1 permits the path $a \rightarrow p \rightarrow q \rightarrow r$, and so supports the statement $a \rightarrow r$; Γ_5 neither permits the path nor supports the statement.

5.2 Soundness

A reasoning mechanism should be sound, at least in the sense that it never leads from a consistent set of assumptions to an inconsistent conclusion. Classical deductive systems, for example, are sound in this sense; but of course, if a reasoner based on classical logic were ever supplied with an inconsistent set of assumptions, it would then support any conclusion at all.

Like classical logic, the inheritance reasoner we describe here is also sound in this standard sense. However, as we mentioned earlier, in Section 3.2.1, it is designed to behave more sensibly than classical logic in the presence of conflicting information. Although this reasoner will conclude from a network containing conflicting statements every statement actually contained in that network, the effect of these conflicts is localized: they don't lead the reasoner to other, possibly irrelevant conclusions. The following theorem shows that the reasoner we describe won't ever draw conflicting conclusions from a network unless that network already contains conflicting statements, as conflicting direct links; and even if the network does happen to contain conflicting statements, our reasoner won't draw any conflicting conclusions that aren't already contained in the network—it will never draw any *new* conflicting conclusions.

Theorem 1 *If Γ supports both $x \rightarrow y$ and $x \not\rightarrow y$, then both $x \rightarrow y \in \Gamma$ and $x \not\rightarrow y \in \Gamma$.*

Proof. Suppose Γ supports both $x \rightarrow y$ and $x \not\rightarrow y$, but doesn't contain both $x \rightarrow y$ and $x \not\rightarrow y$. Then either (i) Γ supports exactly one of these statements only through compound paths, or (ii) Γ supports both of these statements only through compound paths. It is easy to see from clauses II.1.(c) and II.2.(c) of the inheritance definition that (i) is impossible: if Γ contains either statement, it can't permit a compound path enabling the other. We prove the theorem by showing that (ii) is impossible as well.

Suppose (ii) is true. Then Γ permits some path of the form $x \rightarrow \sigma_1 \rightarrow u_1 \rightarrow y$, and also path of the form $x \rightarrow \sigma_2 \rightarrow u_2 \not\rightarrow y$. Since Γ permits a compound path from x to y, it follows from II.1.(a)–(b) and II.2.(a)–(b) that Γ permits some path of the form $x \rightarrow \tau \rightarrow v$ with either $v \rightarrow y \in \Gamma$ or $v \not\rightarrow y \in \Gamma$. Let $x \rightarrow \tau' \rightarrow v'$ be a path of minimal degree satisfying this condition—that is, Γ permits $x \rightarrow \tau' \rightarrow v'$ and either $v' \rightarrow y \in \Gamma$ or $v' \not\rightarrow y \in \Gamma$; and there is no path $x \rightarrow \tau'' \rightarrow v''$ with $\deg_\Gamma(x \rightarrow \tau'' \rightarrow v'') < \deg_\Gamma(x \rightarrow \tau' \rightarrow v')$ such that Γ permits $x \rightarrow \tau'' \rightarrow v''$ and either $v'' \rightarrow y \in \Gamma$ or $v'' \not\rightarrow y \in \Gamma$. Suppose $v' \rightarrow y \in \Gamma$. From the assumption that (ii), it follows that there are no direct links in Γ from x to y. Therefore, since Γ permits $x \rightarrow \sigma_2 \rightarrow u_2 \not\rightarrow y$, it follows from II.2.(d) that there must be a node z such that $\Gamma \models x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v'$ and $z \not\rightarrow y \in \Gamma$. From this it follows that Γ permits $x \rightarrow \tau_2 \rightarrow z$, where $\deg_\Gamma(x \rightarrow \tau_2 \rightarrow z) < \deg_\Gamma(x \rightarrow \tau' \rightarrow v')$ and $z \rightarrow y \in \Gamma$ or $z \not\rightarrow y \in \Gamma$. Therefore, the path $x \rightarrow \tau' \rightarrow v'$ cannot be of minimal degree, contrary to assumption. If $v' \not\rightarrow y \in \Gamma$, it follows likewise from the assumption that Γ permits $x \rightarrow \sigma_1 \rightarrow u_1 \not\rightarrow y$ and II.1.(d) that $x \rightarrow \tau' \rightarrow v'$ cannot be of minimal degree. ∎

5.3 Stability

The property of *stability* is the property of being insensitive in certain ways to redundant information. In general, there are a number of different stability properties we might choose to require in an acceptable reasoner, and the differences between them can be subtle. Just to illustrate the kind of thing that goes wrong when an inheritance reasoner is entirely *un*stable, however, let's consider for a moment a shortest-path reasoning algorithm, such as that suggested by Fahlman [5], which resolves differences among conflicting paths by favoring the shortest.

Consider the nets Γ_{10} and Γ_{11} (Figures 10 and 11). Evidently, Γ_{11} results from Γ_{10} simply through the addition of the link $a \rightarrow r$, which is both atomic and *redundant*—in

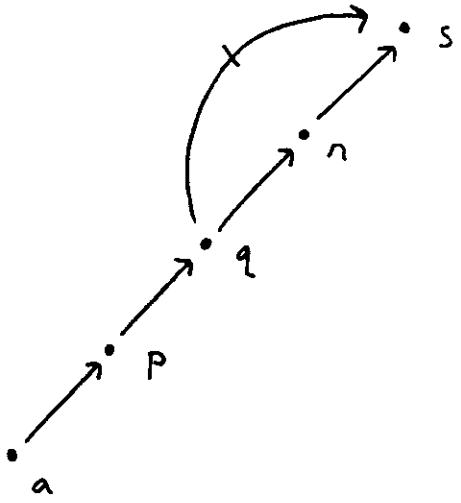

Figure 10: Γ_{10}

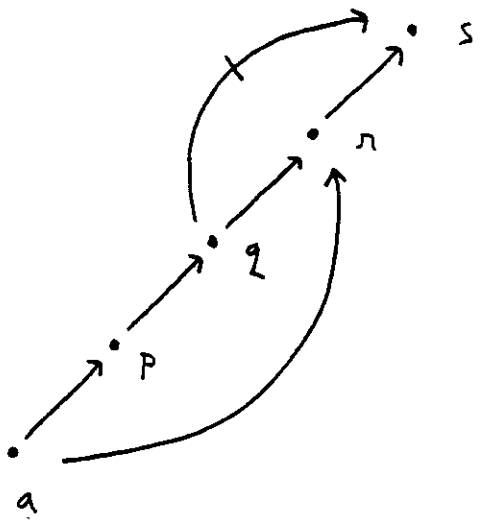

Figure 11: Γ_{11}

the sense that shortest-path reasoning applied to Γ_{10} already tells us that the statement represented by this link is true. Still, even though this atomic link is redundant from the standpoint of Γ_{10}, adding it to Γ_{10} changes the semantics of that net, according to the shortest-path approach to inheritance reasoning: while Γ_{10} supports the statement $a \not\rightarrow s$ (since the path $a \rightarrow p \rightarrow q \not\rightarrow s$ is shorter than the conflicting path $a \rightarrow p \rightarrow q \rightarrow r \rightarrow s$), the new net Γ_{11} would support the statement $a \rightarrow s$ instead (since the path $a \rightarrow r \rightarrow s$ is now shorter than the conflicting path $a \rightarrow p \rightarrow q \not\rightarrow s$).

The example works out differently according to the analysis of inheritance presented here (as well as that of [16]). On the present analysis, it is clear that the two nets Γ_{10} and Γ_{11} support exactly the same statements. In particular, Γ_{11} doesn't now permit the path $a \rightarrow r \rightarrow s$; so, like Γ_{10}, the net Γ_{11} doesn't support the statement $a \rightarrow s$. This situation illustrates the following *atomic stability* theorem, which shows that the reasoner we describe is stable with respect to redundant atomic statements in a way that shortest-path reasoning is not.

Theorem 2 *For an atomic statement A, if Γ supports A, then for any statement B, $\Gamma \cup \{A\}$ supports B if and only if Γ supports B.*

The theorem is an immediate consequence of the following more general lemma, which establishes a connection between the set of paths permitted by a net, and the set of paths permitted once that net is supplemented with a redundant atomic statement.

Lemma 1 *(A) Suppose* $\Gamma \models a \rightarrow \delta \rightarrow p$. *It follows that, if* $\Gamma \models \sigma$, *then* $\Gamma \cup \{a \rightarrow p\} \models \sigma$; *and also that, if* $\Gamma \cup \{a \rightarrow p\} \models \sigma$, *then* $\Gamma \models \sigma^*$, *where* σ^* *is the result of replacing any occurrence of the link* $a \rightarrow p$ *in* σ *by the path* $a \rightarrow \delta \rightarrow p$. *(B) Likewise, suppose* $\Gamma \models a \rightarrow \delta \not\rightarrow p$. *It then follows that, if* $\Gamma \models \sigma$, *then* $\Gamma \cup \{a \not\rightarrow p\} \models \sigma$; *and also, if* $\Gamma \cup \{a \not\rightarrow p\} \models \sigma$, *that* $\Gamma \models \sigma^*$, *where* σ^*, *now, is the result of replacing any occurrence of the link* $a \not\rightarrow p$ *in* σ *by the path* $a \rightarrow \delta \not\rightarrow p$.

Proof. We prove only (A); the proof of (B) is similar. First, suppose σ is a direct link. Let $\Gamma \models \sigma$. Then we know from Case I of the definition that $\sigma \in \Gamma$; so $\sigma \in \Gamma \cup \{a \rightarrow p\}$; so $\Gamma \cup \{a \rightarrow p\} \models \sigma$. Now let $\Gamma \cup \{a \rightarrow p\} \models \sigma$. If $\sigma \neq a \rightarrow p$, then $\sigma \in \Gamma$, and also $\sigma = \sigma^*$; so $\Gamma \models \sigma^*$. If $\sigma = a \rightarrow p$, then $\sigma^* = a \rightarrow \delta \rightarrow p$; so $\Gamma \models \sigma^*$ by assumption.

Next, suppose σ is a compound path, with $\deg_{\Gamma \cup \{a \rightarrow p\}}(\sigma) = n$. As an inductive hypothesis, we suppose that for all σ' with $\deg_{\Gamma \cup \{a \rightarrow p\}}(\sigma') < n$, we know both (i) that $\Gamma \cup \{a \rightarrow p\} \models \sigma'$ if $\Gamma \models \sigma'$ and (ii) that $\Gamma \models \sigma'^*$ if $\Gamma \cup \{a \rightarrow p\} \models \sigma'$. To carry out the inductive step of the proof, we need to consider two subcases, depending on whether σ is a positive or a negative compound path; and for each subcase, it is necessary to show both that $\Gamma \cup \{a \rightarrow p\} \models \sigma$ if $\Gamma \models \sigma$, and that $\Gamma \models \sigma^*$ if $\Gamma \cup \{a \rightarrow p\} \models \sigma$. The cases are largely similar; we show here only that if σ is a positive compound path, of the form $x \rightarrow \sigma_1 \rightarrow u \rightarrow y$, then $\Gamma \models \sigma^*$ whenever $\Gamma \cup \{a \rightarrow p\} \models \sigma$.

Suppose that $\Gamma \cup \{a \rightarrow p\} \models \sigma$. We know from Case II.1 of the definition that

(a) $\Gamma \cup \{a \rightarrow p\} \models x \rightarrow \sigma_1 \rightarrow u$,

(b) $u \rightarrow y \in \Gamma \cup \{a \rightarrow p\}$,

(c) $x \not\rightarrow y \notin \Gamma \cup \{a \rightarrow p\}$,

(d) For all v such that $\Gamma \cup \{a \rightarrow p\} \models x \rightarrow \tau \rightarrow v$ with $v \not\rightarrow y \in \Gamma \cup \{a \rightarrow p\}$, there exists z such that $z \rightarrow y \in \Gamma \cup \{a \rightarrow p\}$ and either $z = x$ or $\Gamma \cup \{a \rightarrow p\} \models x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v$.

From (a) and (ii) of the inductive hypothesis, we can conclude that

(a*) $\Gamma \models (x \rightarrow \sigma_1 \rightarrow u)^*$.

We know that $u \rightarrow y \neq a \rightarrow p$, since a is an individual and u must be a kind; so we can conclude from (b) that

(b*) $u \rightarrow y \in \Gamma$.

It follows directly from (c) that

(c*) $x \not\rightarrow y \notin \Gamma$.

Finally, suppose that, for some v, $\Gamma \models x \rightarrow \tau \rightarrow v$ with $v \not\rightarrow y \in \Gamma$. Of course, $v \not\rightarrow y \in \Gamma \cup \{a \rightarrow p\}$ as well, and we know from (i) of the inductive hypothesis that $\Gamma \cup \{a \rightarrow p\} \models x \rightarrow \tau \rightarrow v$. Therefore (d) above tells us that there exists z such that $z \rightarrow y \in \Gamma \cup \{a \rightarrow p\}$ and either $z = x$ or $\Gamma \cup \{a \rightarrow p\} \models x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v$. Again, though, since z must be a kind, we know that $z \rightarrow y \neq a \rightarrow p$, and so we have $z \rightarrow y \in \Gamma$. If $\Gamma \cup \{a \rightarrow p\} \models x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v$, then we can conclude from (ii) of the inductive hypothesis that $\Gamma \models (x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v)^*$. Thus, we have

(d*) For all v such that $\Gamma \models x \rightarrow \tau \rightarrow v$ with $v \not\rightarrow y \in \Gamma$, there exists z such that $z \rightarrow y \in \Gamma$ and either $z = x$ or $\Gamma \models (x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v)^*$.

And from (a*) through (d*), we can conclude that $\Gamma \models \sigma^*$. ∎

What Theorem 2 shows, again, is that you can't affect the set of statements supported by a net by supplementing it with atomic statements it already supports. The analog to this theorem fails, however, when a net is supplemented with a redundant *generic* statement. To see this, consider the nets Γ_{12} and Γ_{13} (Figures 12 and 13). Here, Γ_{12} doesn't permit the path $a \rightarrow p \rightarrow q \rightarrow r \rightarrow s$ (since its initial segment $a \rightarrow p \rightarrow q \rightarrow r$ is preempted); so this net doesn't support the statement $a \rightarrow s$. On the other hand, since Γ_{12} permits the path $q \rightarrow r \rightarrow s$, it does supports the generic statement $q \rightarrow s$. Evidently, Γ_{13} results from Γ_{12} only through the addition of this statement, redundant from the standpoint of Γ_{12}. Yet, Γ_{13} *does* now support the statement $a \rightarrow s$, since Γ_{13} permits $a \rightarrow p \rightarrow q \rightarrow s$.

It's hard to know what to make of examples like this. We view it almost as a criterion of acceptability in an inheritance reasoner that it should exhibit *atomic* stability. No one has ever produced a counterexample to atomic stability with any intuitive force; and it was, in part, the failure of this property in shortest-path reasoners that motivated both the theory of Touretzky [16] and the present analysis. When it comes to *generic* stability, however, the matter is more complicated.

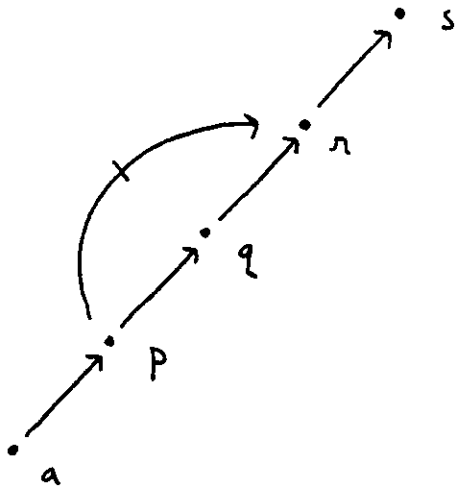

Figure 12: Γ_{12}

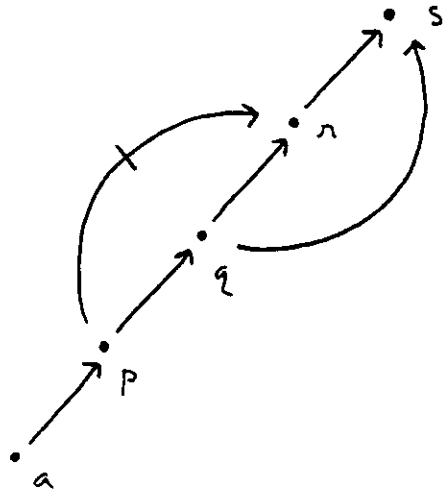

Figure 13: Γ_{13}

On one hand, it is surprising—at least from the standpoint of our analogy between inheritance reasoning and deductive reasoning—to find that an acceptable inheritance reasoner might fail to exhibit generic stability. The deductive analog of a network is a set of hypotheses, or axioms; the statements supported by a network are like the theorems derivable from those axioms. From the standpoint of our analogy, then, the failure of stability is like a situation in which the consequences of a set of axioms would be affected if the axioms were supplemented, not just with an arbitrary statement, but with a *theorem derivable from those axioms*—and this makes little sense even from a general deductive point of view, allowing for the possibility of nonmonotonic deductive systems.[4] On the other hand, even though it indicates a break in our analogy between inheritance and deduction, it is possible to argue that the kind of generic instability exhibited above might actually turn out to be a *desirable* property in an inheritance reasoner: it suggests a way in which the graph-theoretic nature of inheritance reasoning allows a kind of sensitivity to the structure of arguments that is difficult to achieve in deductive systems.

To illustrate, we supply Γ_{12} and Γ_{13} with the following interpretation, inspired by an example from Sandewall [14]: a = Moby, p = Whales, q = Mammals, r = Land-dwellers, s = Air-breathers. On this interpretation, what Γ_{12} tells us directly is that Moby is a whale, that whales are mammals, that mammals are land-dwellers, that whales aren't

[4]In a deductive system with consequence relation $\vdash$, nonmonotonicity is the principle that if $\Gamma \vdash A$ then $\Gamma \cup \Delta \vdash A$; stability is the principle that if $\Gamma \vdash A$ then $\Gamma \cup \{A\} \vdash B$ iff $\Gamma \vdash B$. Even nonmonotonic consequence relations—such as the relation $\vdash_P$ defined by McCarthy [9]—tend to be stable.

land-dwellers, and that land-dwellers are air-breathers. Given just this information, we shouldn't be able to conclude that Moby is an air-breather: only land-dwellers are known directly to be air-breathers, and we can conclude that Moby isn't a land-dweller, since he is a whale. Of course, Γ_{12} does support the conclusion that Moby is a mammal, and also the conclusion that mammals are air-breathers. But we can't put these two ideas together in Γ_{12} to conclude that Moby is an air-breather. In Γ_{12}, the argument showing that mammals are air-breathers depends on their being land-dwellers. Therefore, we shouldn't be able to apply this general conclusion about mammals to Moby, since the general conclusion holds of mammals only in virtue of their being land-dwellers, and we know of Moby in particular that he is not a land-dweller. It is different in Γ_{13}. Here, the fact that mammals are air-breathers no longer depends solely on the fact that they are land-dwellers. According to Γ_{13}, mammals would be air-breathers even if they weren't land-dwellers. Therefore, the fact that Moby in particular isn't a land-dweller shouldn't interfere in Γ_{13} with the general argument that, since he is a mammal, he is an air-breather.

5.4 Intersections of credulous extensions

We mentioned in Section 3 that the credulous (or belief-hungry) approach tends to associate with nets containing compound conflicting paths a number of different consistent extensions, or fixed points. It is tempting, therefore, to suppose that the set of paths permitted by a given net under the present skeptical analysis might simply be the *intersection* of the various extensions associated with that net according to the credulous analysis provided by [16]. However, nets like Γ_{14}—which have the topology of *nested* Nixon Diamonds—show that this is not so. In this case, we have $\Gamma_{14} \models a \rightarrow p \not\rightarrow q$. The potentially conflicting path $a \rightarrow s \rightarrow t \rightarrow q$ poses no problem; this path is not permitted, since its initial segment $a \rightarrow s \rightarrow t$ is itself neutralized by the path $a \rightarrow r \not\rightarrow t$. But the path $a \rightarrow p \not\rightarrow q$ isn't contained in all the credulous extensions associated with this net; some contain instead the path $a \rightarrow s \rightarrow t \rightarrow q$.

5.5 Decoupling

In inheritance reasoners that construct inference paths through backward chaining, such as that of Touretzky [16], conclusions about items in a network depend on conclusions about their immediate superiors. According to Touretzky's theory, for example, a path of the form $x \rightarrow u \rightarrow \sigma \rightarrow y$ will belong to an extension only if the path $u \rightarrow \sigma \rightarrow y$ also

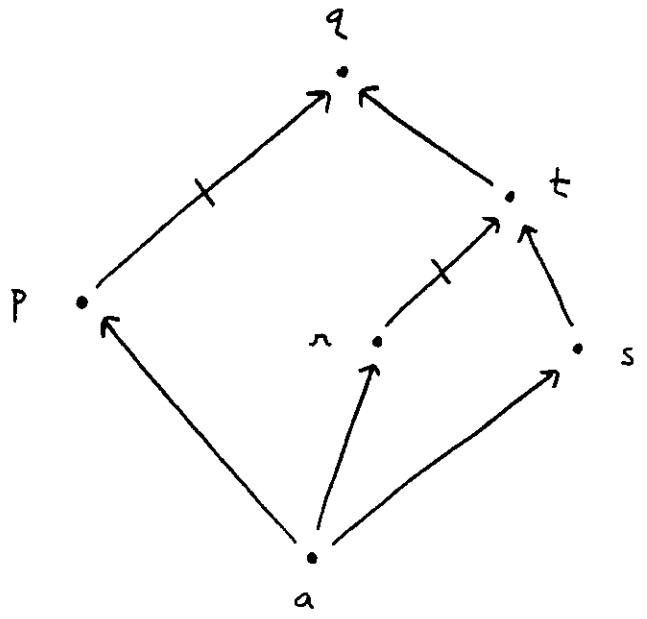

Figure 14: Γ_{14}

belongs to that extension; a path of the form $x \rightarrow u \rightarrow \sigma \not\rightarrow y$ will belong to an extension only if $u \rightarrow \sigma \not\rightarrow y$ does.

As the net Γ_{15} shows, our analysis allows items in a network to be *decoupled* from their immediate superiors, in the sense that it allows particular items to possess properties possessed by none of their immediate superiors. Here, we have $\Gamma_{15} \models a \rightarrow p \rightarrow q \rightarrow s$. The potentially conflicting path $a \rightarrow p \rightarrow r \not\rightarrow s$ poses no problem since its compound initial segment $a \rightarrow p \rightarrow r$ conflicts with the direct link $a \not\rightarrow r$. On the other hand, though Γ_{15} permits $a \rightarrow p \rightarrow q \rightarrow s$, and so supports the statement $a \rightarrow s$, the net does not permit the path $p \rightarrow q \rightarrow s$, and indeed does not support the statement $p \rightarrow s$.

This kind of decoupling can seem a bit anomalous if one's ideas about inheritance reasoning are conditioned by the top-down or "property flow" approach, according to which individuals are supposed to inherit their properties strictly in virtue of belonging to certain classes of things—their ancestors in the network—which possess those properties. The problem is that, while Γ_{15} supports the statement that the individual a is an s, it is unclear how a could have *inherited* this property. After all, the only immediate ancestor of a in the network is the node p. According to the top-down approach, then, a must have inherited all the positive properties it does inherit simply in virtue of being a p; if it possess any particular property, such as being an s, this could only be due to the fact that p's possess that property. But as we have seen, Γ_{15} *doesn't* support the statement that p's are s's.

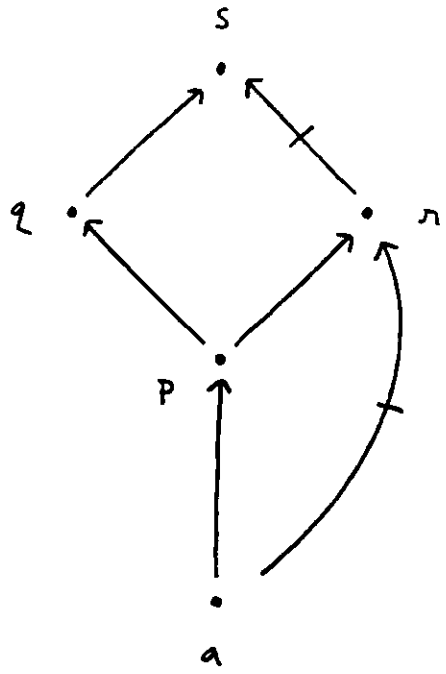

Figure 15: Γ_{15}

Against the background of the bottom-up or "argument construction" view of inheritance reasoning, however, the situation presented by this example is perfectly coherent. Since Γ_{15} contains the materials for constructing unpreempted, compound arguments enabling both the conclusion that p's are s's and the conclusion that p's are not s's, our broadly skeptical point of view forces us to withhold judgment, endorsing neither of these conclusions. The individual a, though, is a *particular* p for which the general kind of argument enabling the conclusion that p's are not s's is blocked: that argument depends on the information that p's are r's, but Γ_{15} tells us explicitly that a is not an r. Since the general argument that p's are not s's is explicitly blocked for this particular individual, then, it cannot conflict in the case of a with the argument that p's are s's; so we conclude that a is an s.

6 A hybrid inference algorithm

Inheritance networks are attractive as formalisms for knowledge representation because they allow information to be organize in such a way that certain important kinds of inference can be carried out through efficient graph-searching techniques. In particular, ever since their inception in the work of Quillian [11], network representations have been associated with *parallel* inference schemes. This tendency culminated in the NETL system of Fahlman [5], which combined a nonmonotonic network representation language with a

massively parallel reasoning architecture, known as a Parallel Marker Propagation Machine (PMPM). Unfortunately, the NETL representation language was never provided with a clear semantics of its own, independent of the associated inference algorithms; and it turned out, because of their treatment of exceptions, that these algorithms often led to anomalous results [7]. Once a satisfactory semantic account was developed for inheritance networks with exceptions—initially, with the work of Etherington [4] and Touretzky [16]—it soon became clear that the kinds of inferences appropriate to these networks could not be carried out through purely parallel reasoning.

In this section, after reviewing Fahlman's PMPM architecture, we present a *hybrid* (parallel-serial) inference algorithm that reasons in accord with the definition of inheritance presented here. The algorithm is designed to exploit the parallelism of the PMPM to the greatest extent possible, resorting to serial reasoning only when necessary.

6.1 Parallel Marker Propagation Machines

A PMPM is an automaton composed of active elements that play the roles of nodes and links in a graph. Each element has a small number of internal states (marker bits, which can be on or off, representing the presence or absence of markers), and a limited ability to communicate information to the elements to which it is connected. The nodes and links in a PMPM are responsive to various marker propagation commands, each of which directs the assignment of markers to particular nodes, often by "propagating" them from one node to another through the intervening links. PMPM's are SIMD (Single Instruction stream, Multiple Data stream) machines: marker propagation commands are broadcast globally to all elements and executed in parallel by the elements to which they apply. Parallel marker propagation algorithms can be described as sequences of marker propagation commands; the result of executing such an algorithm in a particular net is a *coloring*—a static assignment of marker bits to nodes—that is used to convey some information about the net.

The notation used here for specifying marker propagation commands is a slight extension of the one defined in Touretzky [16]. Commands may be either conditional or unconditional. Unconditional commands are executed by all elements regardless of their current state. The unconditional command **clear**[M_1], for instance, causes all elements to clear marker bit M_1. Conditional commands are more common. The command

link-type[“→”], **on-tail**[M_1], **off-head**[M_1] $\Longrightarrow$ **set-head**[M_1]

would be executed by any element meeting the conditions on the left hand side of the arrow: if the element represents a link of type “→”, the node at its tail bears the marker M_1, and the node at its head does not bear marker M_1, then the link will perform the action specified on the right hand side of the arrow, marking the node at its head with M_1.

Looping is accomplished with a simple **loop** *body* **endloop** construct, which repeats the commands in the body of the loop until no conditional command appearing in the body has its left hand side satisfied. The following loop, for example, propagates the marker M_1 up “→” links, thereby computing the transitive closure of the “→” relation. The loop terminates when all eligible nodes have been marked with M_1. (The **off-head**[M_1] condition assures that nodes already marked with M_1 are not eligible to be marked on subsequent iterations.)

loop
 link-type[“→”], **on-tail**[M_1], **off-head**[M_1] $\Longrightarrow$ **set-head**[M_1]
endloop

It is also possible to address particular nodes by name using conditional commands. The node x would be selected by placing the restriction **name**[x] on the left hand side of the conditional arrow; only the element representing that node would then respond. This technique is used to select and mark an initial node at the beginning of certain marker propagation procedures. For example, the procedure below would compute the transitive closure of the “→” relation starting at a given node x, marking the nodes in the resulting set with M_1.

procedure transitive-closure(x: **node**) = **begin**
 clear[M_1];
 name[x] $\Longrightarrow$ **set**[M_1];
 loop
  link-type[“→”], **on-tail**[M_1], **off-head**[M_1] $\Longrightarrow$ **set-head**[M_1]
 endloop
end

The set of conditions on the left hand side of the conditional arrow are always treated conjunctively. Many conditions accept multiple arguments, which also are treated conjunctively; for example, **on-tail**[M_1, M_2] is satisfied by a link element only if it satisfies both **on-tail**[M_1] and **on-tail**[M_2]. It is possible, however, to specify disjunctions of markers on the left hand side, by using a different set of multi-argument commands: the **link-type** condition and all conditions beginning with "**any-**" are disjunctive. For example, the command **any-on-tail**[M_1,M_2] calls those link elements that satisfy either **on-tail**[M_1] or **on-tail**[M_2].

A parallel marker propagation machine is controlled by a host computer that broadcasts commands to all the individual elements, where they are executed in synchrony. The extension we have made here to the PMPM algorithm notation—a loop of the form **for** *var* **in** <*parallel-condition*> **do** *body*—doesn't really affect the parallel capabilities of the machine at all, but involves only statements to be executed by the host. The point of the extension is to allow elements to be processed serially when necessary. The *parallel-condition* specifies a conditional test, equivalent to the left hand side of a rule, that is broadcast to all elements. Elements satisfying the condition are then identified by the host computer, using some addressing mechanism which we won't go into here (see Fahlman [5] for details), and processed serially.

6.2 Skeptical inheritance on a PMPM

Because each of the computing elements of a PMPM can represent only a small, fixed number of marker bits, it isn't possible to use a PMPM algorithm to compute, all at once, the entire theory of a net: for a net with N nodes, that would require each node to carry at least $2N$ marker bits. Moreover, since it is only under rare circumstances that a user would actually be interested in knowing the entire theory of a network, it would be unwise in any case to invest the computational resources necessary for computing the full theory. In the general case, the user comes to a database with a particular query in mind: he wants to know whether the database supports a particular statement, or its negation. Therefore, what we define here is a query procedure—*query*(x, y)—which is able to determine, for any net Γ and nodes x and y, whether Γ supports the statement $x \rightarrow y$, the statement $x \not\rightarrow y$, neither statement, or both.

The procedure we define is economical. It scans only that portion of the network directly relevant to a particular query. It uses only fourteen markers all told, and only the two

markers M_T and M_F to represent the result of the query. As a result of performing the *query*(x, y) in the net Γ, the marker M_T will be present on the node y iff Γ supports $x \rightarrow y$, and marker M_F will will be present on the node y iff Γ supports $x \not\rightarrow y$.

6.2.1 Degree^{x,y}

In Section 4, we ordered the paths in a given net by degree, and then proceeded to define the permission relation through an induction on the degree of a path in a net. In that context—where definition, not implementation, was the issue—the ordering by degree was appropriate, since it is a particularly simple ordering, and it is adequate, in the sense that all the paths that could possibly be relevant to a given path are assigned a lesser degree.

In the present context, however, since implementation is itself the issue, the ordering by degree is no longer quite so appropriate. For reasons of efficiency, we would like our query procedure to examine the minimum number of paths necessary to decide whether a particular statement is supported by a net; but if it were to sort through the paths in the net by degree, in addition to all the paths relevant to the query, the query procedure would wind up considering a number of irrelevant paths as well. For example, in the net Γ_{16} (Figure 16), both the paths $x \rightarrow t \rightarrow u$ and the path $x \rightarrow p \rightarrow q \not\rightarrow r$ are assigned a lower degree than the path $x \rightarrow z \rightarrow y$. Since it is obvious from the structure of the network, however, that neither of these paths could possibly have an effect on the path $x \rightarrow z \rightarrow y$, it would be wasteful for the procedure *query*(x, y) even to consider them. In order to implement our query procedure efficiently, we need to define another ordering of the paths in a net, which differs from the standard ordering by degree in considering only the paths relevant to a particular query.

As a first step, we specify the *restriction* of a net Γ with respect to the *query*(x, y). Intuitively, this query-restricted network—written, $\Gamma^{x,y}$—is supposed to represent the subgraph of Γ which it is necessary to examine in order to determine whether Γ supports either of the statements $x \rightarrow y$ or $x \not\rightarrow y$. We capture the notion formally by specifying that $\Gamma^{x,y}$ is the minimal set containing (i) every link on every path in Γ from x to y, as well as (ii) every link on every path in Γ from x to w, for all nodes w occurring in $\Gamma^{x,y}$. For example, we have $\Gamma_{16}^{x,y} = \{x \rightarrow z, z \rightarrow y\}$, since $x \rightarrow z \rightarrow y$ is the only path in Γ_{16} from x to y, and there are no other paths in Γ_{16} from x to any of the nodes on this path. However, $\Gamma_{17}^{x,y}$ includes, as it should, every link in Γ_{17} except for the two links $x \rightarrow t$ and $t \rightarrow u$. Since Γ_{17} contains the additional path $x \rightarrow v \rightarrow r \not\rightarrow y$, the path $x \rightarrow p \rightarrow q \not\rightarrow r$, which was

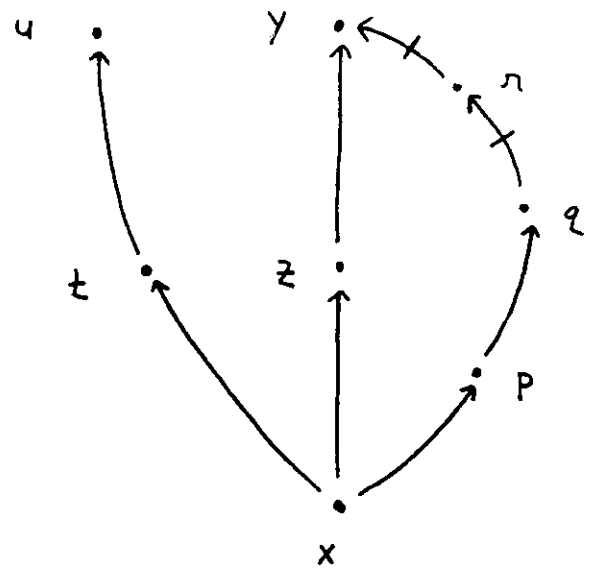

Figure 16: Γ_{16}

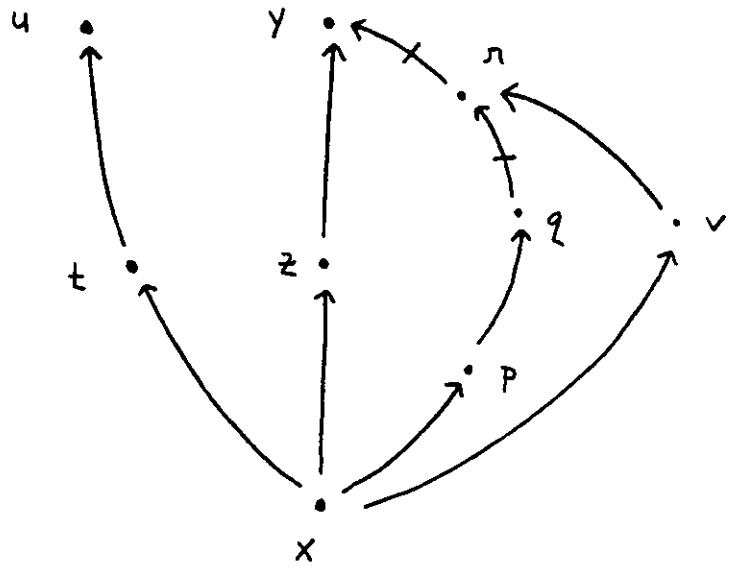

Figure 17: Γ_{17}

contained in Γ_{16} but irrelevant there to *query(x,y)*, is now relevant to the query; so the links in this path must be contained in $\Gamma_{17}^{x,y}$.

Once we have defined the query-restricted net $\Gamma^{x,y}$ in this way, it is easy to show that, for any path σ in $\Gamma^{x,y}$, $\Gamma \models \sigma$ iff $\Gamma^{x,y} \models \sigma$. It is then natural to define the degree of a path σ in a net Γ *with respect to the query(x,y)*—written, $\deg_{\Gamma}^{x,y}(\sigma)$—as the degree of the path σ in the suitably restricted net $\Gamma^{x,y}$. Formally, we take

$$\deg_{\Gamma}^{x,y}(\sigma) = \begin{cases} \deg_{\Gamma^{x,y}}(\sigma) & \text{if } \sigma \text{ is a path in } \Gamma^{x,y} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In order to distinguish it from the familiar (Section 4.1) notion of degree, we refer to this new, query-restricted notion as *degree*^{x,y}. Evidently, the procedure *query(x,y)* can accurately determine whether a net Γ supports $x \rightarrow y$ or $x \not\rightarrow y$ while limiting its attention to $\Gamma^{x,y}$: it need only consider those paths in a net whose degree^{x,y} is defined, and it can consider those paths in order of increasing degree^{x,y}.

For convenience, we will also refer in what follows to the degree^{x,y} of a *node*; where w is a node, the degree^{x,y} in Γ of w is defined as the degree^{x,y} of any path in Γ from x to w. (Of course, all such paths have the same degree^{x,y}).

6.2.2 The algorithm

We can now describe *query(x,y)*, our PMPM algorithm for responding to queries in accord with the definition of skeptical inheritance presented here. We begin with the two subprocedures *trim-for-query(x,y)* and *select-next-degree()*; after that, we present the bulk of *query(x,y)* itself, with the exception of the subprocedure *check-preemption(c)*, which is described last of all.

The procedure *trim-for-query(x, y)* trims the network Γ so that only the query-restricted $\Gamma^{x,y}$ will be considered in future processing. It begins in lines 3 and 4 by marking x with M_x and y with M_y. In the loop at lines 5–7, it propagates the marker M_1 up positive paths in Γ from x. If there is a path in Γ, positive or negative, from x to y, it marks y with M_2 in lines 8-9. Finally, in the loop at lines 10–13, the procedure propagates M_2 down both "$\rightarrow$" links and "$\not\rightarrow$"links in Γ into those nodes already marked with M_1.

```
1  procedure trim-for-query(x, y: node) = begin
2    clear[M_x,M_y,M_1,M_2];
3    name[x]    ⟹   set[M_x];
4    name[y]    ⟹   set[M_y];
5    loop
6     link-type["→"], any-on-tail[M_x,M_1], off-head[M_1]    ⟹   set-head[M_1]
7    endloop
8    link-type["→","↛"], on-tail[M_1], on-head[M_y]
9         ⟹   set-head[M_2];
10   loop
11    link-type["→","↛"], on-head[M_2], on-tail[M_1], off-tail[M_2]
12         ⟹   set-tail[M_2];
13   endloop
14 end
```

The overall effect of carrying out this procedure in a net Γ is to mark each node in $\Gamma^{x,y}$ (with the exception of x itself) with the marker M_2, a fact recorded in the following lemma. In future processing, we then manage to ignore irrelevant paths by restricting certain commands so that they apply only to nodes marked with M_2.

Lemma 2 *As a result of running the procedure* trim-for-query(x, y) *in* Γ, *each node* w *occurring in* Γ *will be marked with* M_2 *iff* $w \neq x$ *and* w *occurs in* $\Gamma^{x,y}$.

Proof. *Trim-for-query(x, y)* begins in lines 5–7 by marking the nodes on positive paths in Γ from x, except for x itself, with M_1. It is clear that y itself will be marked with M_2 in lines 8–9 iff there is a path from x to y in Γ. Now suppose that not all nodes occurring in $\Gamma^{x,y}$ are marked with M_2; in particular, let w be a node of maximal degree^{x,y} in $\Gamma^{x,y}$ that is not marked with M_2. Since w occurs in $\Gamma^{x,y}$, Γ must contain a path either of the form $x \to \tau \to w \to z$ or of the form $x \to \tau \to w \not\to z$, such that z occurs in $\Gamma^{x,y}$; since w is maximal, z will be marked with M_2. Since w is on a positive path from x, w will have to be marked with M_1; so w will be marked with M_2 in the loop at lines 10–13, which contradicts our assumption. Thus, all nodes occurring in $\Gamma^{x,y}$ will be marked with M_2. Further, since only nodes that are on paths from x to a node with M_2 will be marked with M_2, all nodes marked with M_2 will occur in $\Gamma^{x,y}$. ∎

The procedure *select-next-degree()* will be called each time through the main loop of the procedure *query(x,y)* to mark the nodes of degree^{x,y} n.

```
1  procedure select-next-degree() = begin
2    clear[M_deg];
3    link-type["→","↛"], any-on-tail[M_x,M_old], on-head[M_2], off-head[M_old]
4        ⟹  set-head[M_deg];
5    link-type["→","↛"], on-head[M_deg], on-tail[M_2], off-tail[M_x,M_old]
6        ⟹  clear-head[M_deg];
7    on[M_deg]    ⟹  set[M_old];
8  end
```

The effect of this procedure, in the context in which it occurs in *query(x,y)*, is recorded in the following lemma.

Lemma 3 *After executing* trim-for-query(x, y) *and clearing* M_{old} *in a network* Γ, *the n-th call to* select-next-degree() *leaves exactly the nodes of degree*^{x,y} n *in* Γ *marked with* M_{deg}, *and exactly the nodes of degree*^{x,y} *less than or equal to* n *in* Γ *marked with* M_{old}.

Proof. Nodes of degree^{x,y} 1 will be connected to x in $\Gamma^{x,y}$ by direct links, and only direct links. It is obvious that the first time *select-next-degree()* is called, it will mark all such nodes with M_{deg} in lines 3–4; the command on lines 5–6 can have no effect, since it applies only to links not emanating from x; and then at line 7, the nodes marked with M_{deg} will be marked also with M_{old}. Thus the theorem is true for $n = 1$.

For induction, suppose the theorem holds for all calls prior to the n-th. The nodes of degree^{x,y} n are just those nodes w for which (i) $\Gamma^{x,y}$ contains either of the links $u \rightarrow w$ or $u \not\rightarrow w$ with u a node of degree^{x,y} $n-1$; and (ii) $\Gamma^{x,y}$ contains no links $v \rightarrow y$ or $v \not\rightarrow y$ with v a node of degree^{x,y} greater than or equal to n. Given the inductive hypothesis, then, the procedure *select-next-degree()* behaves as follows on its n-th call. First, in line 2, it clears the marker M_{deg} from nodes of degree^{x,y} n. Next, in lines 3–4, it places M_{deg} on exactly those nodes w satisfying (i)—since, by hypotheses, any such node u will already be marked with M_{old}, w will of course be marked with M_2, but w will not yet be marked with M_{old}. Then, in lines 5–6, it clears M_{deg} from all those nodes w satisfying (i) except those also satisfying (ii)—since, if w fails to satisfy (ii), any such node v will of course be marked with M_2 and will by hypothesis be marked neither with M_x nor M_{old}. Finally, in

line 7, the procedure marks the nodes displaying M_{deg} also with M_{old}. Thus, if the theorem is true for the all calls prior to the n-th, it is true also for the n-th call. ∎

At this point, we can describe *query(x, y)* itself, our main inference algorithm.

```
1  procedure query(x,y: node) = begin
2    clear[Mold,MT,MF,MSK,Mtt,Mff];
3    trim-for-query(x,y);
4    loop
5      select-next-degree();
6      link-type["→"], on-tail[Mx], on-head[Mdeg]    ⟹  set-head[MT];
7      link-type["↛"], on-tail[Mx], on-head[Mdeg]    ⟹  set-head[MF];
8      link-type["→"], on-tail[MT], on-head[Mdeg], off-head[MT,MF]
9          ⟹  set-head[Mtt];
10     link-type["↛"], on-tail[MT], on-head[Mdeg], off-head[MT,MF]
11         ⟹  set-head[Mff];
12     on[Mtt], off[Mff]    ⟹  set[MT];
13     on[Mff], off[Mtt]    ⟹  set[MF];
14     for c in <on[Mtt,Mff,Mdeg]> do check-preemption(c);
15   endloop
16 end
```

The procedure begins on line 2 by clearing the result marks M_T, M_F, and M_{SK}, as well as the auxiliary marks M_{tt}, and M_{ff}, and then calling *trim-for-query(x, y)* on line 3 to mark all the relevant nodes with M_2. Then the main loop (lines 4–15) is entered. Through the calls to *select-next-degree()* on line 5, nodes are processed in order of increasing degree: during each iteration n of the loop, the nodes of exactly degree^{x,y} n are selected and marked for processing with M_{deg}. If there are direct links from x to w, where w is a node of degree^{x,y} n, lines 6–7 mark w with M_T or M_F, as appropriate. The real work begins on lines 8–9. Here, each node w of appropriate degree^{x,y} is given the auxiliary marker M_{tt} if it is connected by a "→" link to some node u already marked with M_T. Likewise, in lines 10–11, w is marked with M_{ff} if it is connected by a "↛" link to a node v already marked with M_T. These auxiliary markers indicate tentative conclusions. If w is marked with M_{tt} in lines 8–9, but not with M_{ff} in lines 10–11, there is tentative evidence for thinking that the statement $x \rightarrow w$ should be supported, and no evidence to the contrary; therefore, the node will be marked with M_T in line 12. Likewise, if w is marked with M_{ff} but not M_{tt}, it

will be marked in line 13 with M_F. However, some nodes may be marked with both M_{tt} and M_{ff}: if w is such a node there is tentative reason to think $x \rightarrow w$ is supported, as well as a tentative reason for thinking that $x \not\rightarrow w$ is supported. For each of these "conflicted" nodes, we must then call the following *check-preemption* procedure individually, in line 14, to determine whether either of the conflicting tentative arguments are preempted.

```
1  procedure check-preemption(c: node) = begin
2    clear[M_c,M_dir,M_pre];
3    name[c]    ⟹  set[M_c], clear[M_tt,M_ff];
4    link-type["→","↛"], on-tail[M_T], on-head[M_c]    ⟹  set-tail[M_dir];
5    loop
6      link-type["→"], any-on-tail[M_dir,M_pre], on-head[M_T],
7        off-head[M_pre,M_F]    ⟹  set-head[M_pre];
8    endloop;
9    link-type["→"], on-tail[M_dir], off-tail[M_pre]
10       ⟹  set-head[M_tt];
11   link-type["↛"], on-tail[M_dir], off-tail[M_pre]
12       ⟹  set-head[M_ff];
13   on[M_c], on[M_tt], off[M_ff]    ⟹  set[M_T];
14   on[M_c], on[M_ff], off[M_tt]    ⟹  set[M_F];
15   on[M_c], on[M_tt,M_ff]    ⟹  set[M_SK];
16 end
```

Where c is such a conflicted node, the procedure *check-preemption(c)* marks nodes with direct links to c with M_{dir} on line 4. It then marks preempted nodes, in the loop at lines 5–8, by propagating M_{pre} upwards from nodes with M_{dir}. In lines 9–12, the tentative markers M_{tt} and M_{ff}, which were deleted from c in line 3, are reset only if they were propagated from nodes which are not marked as preempted. If only one of these two tentative markers is reset, it must be that the tentative conflicting evidence was preempted; so c is marked in lines 13–14 either with M_T or with M_F, as appropriate. If both of the tentative markers are reset, then neither of the conflicting tentative arguments is preempted. In accord with our general skeptical viewpoint, therefore, these paths neutralize each other; so the node c is marked in line 15 with M_{SK}.

6.3 Correctness of the algorithm

The PMPM algorithm we have described in this section can be proved to be correct—that is, both sound and complete—with respect to the inheritance definition from Section 4.

Theorem 3 *As a result of executing the procedure* query(x, y) *in a net* Γ, *the node* y *will be marked with* $\mathrm{M_T}$ *iff* Γ *supports* $x \rightarrow y$, *and with* $\mathrm{M_F}$ *iff* Γ *supports* $x \not\rightarrow y$.

This theorem follows at once from the following lemma, along with the the observation that no command in *query(x,y)* ever deletes either of the markers $\mathrm{M_T}$ or $\mathrm{M_F}$.

Lemma 4 *Suppose procedure* query(x, y) *is executed in* Γ; *let* w *be a node of degree*^{x,y} n *in* Γ. *Then* w *will be marked with* $\mathrm{M_T}$ *during the* n*-th iteration of the main loop in* query(x, y) *iff* Γ *supports* $x \rightarrow w$, *and* w *will be marked with* $\mathrm{M_F}$ *during the* n*-th iteration of the main loop in* query(x, y) *iff* Γ *supports* $x \not\rightarrow w$.

Proof. First, let the degree^{x,y} of w be 1, so that Γ contains direct links, and only direct links, from x to w. Suppose $x \rightarrow w \in \Gamma$. Then of course Γ supports $x \rightarrow w$. Lemma 3 tells us that, on the first time through its main loop, *query(x,y)* marks w with $\mathrm{M_{deg}}$ in line 5; then w is marked in line 6 with $\mathrm{M_T}$. Similar reasoning shows us that the theorem holds also when $x \not\rightarrow w \in \Gamma$. Therefore, the theorem is true when w is a node of degree^{x,y} 1. Assuming the theorem is true for nodes of degree^{x,y} less than n, we show that it's true also for the node w, where w is of degree^{x,y} n. The induction proceeds in four parts.

Part 1: completeness for positive paths. Suppose Γ supports $x \rightarrow w$. If $x \rightarrow w \in \Gamma$, Lemma 3 tells us that w will be marked with $\mathrm{M_{deg}}$ by the time the n-th pass through the main loop of *query(x,y)* reaches line 6; it will therefore be marked with $\mathrm{M_T}$ in line 6. If $x \rightarrow w \notin \Gamma$, then Γ must permit some path of the form $x \rightarrow \sigma_1 \rightarrow u \rightarrow w$. By II.1.(a) of the inheritance definition, we know $\Gamma \models x \rightarrow \sigma_1 \rightarrow u$; so by inductive hypothesis, *query(x,y)* marks u with $\mathrm{M_T}$ prior to the n-th iteration of the main loop. By II.1.(b), we know $u \rightarrow w \in \Gamma$. Therefore, on the n-th iteration of the main loop, when Lemma 3 tells us that the node w is marked with $\mathrm{M_{deg}}$, w will be marked in lines 8–9 with $\mathrm{M_{tt}}$. If there is no node v such that $v \not\rightarrow w \in \Gamma$ and $\Gamma \models x \rightarrow \tau \rightarrow v$, we can see using the inductive hypothesis that the node w cannot be marked with $\mathrm{M_{ff}}$ in lines 10–11. It will therefore be marked with $\mathrm{M_T}$ at line 12. On the other hand, if there is a node v such that $v \not\rightarrow w \in \Gamma$ and $\Gamma \models x \rightarrow \tau \rightarrow v$, we can see using the inductive hypothesis that the node w will be

marked with M_{ff} in lines 10–11; it will therefore satisfy the condition in the **for** loop at line 14, and so *check-preemption(c)* will have to be called, with $c = w$.

For any such node v, II.1.(d) tells us that there must be a z such that $z \rightarrow w \in \Gamma$ and either $z = x$ or $\Gamma \models x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v$. There may, of course, be many nodes z satisfying this condition: let z' be be one of minimal degree^{x,y}. In line 3 of *check-preemption(c)*, the markers M_{tt} and M_{ff} are cleared from w, and M_c is set. In line 4, the marker M_{dir}—indicating a direct link to w—is set on both the nodes v and z'. Since $\Gamma \models x \rightarrow \tau_1 \rightarrow z' \rightarrow \tau_2 \rightarrow v$, the inductive hypothesis tells us that every nodes on this path must display the marker M_T by the time we enter the n-th iteration of the main *query(x,y)* loop. Therefore, the loop in lines 5–8 of *check-preemption(c)* will propagate M_{pre} up this path, beginning with the node immediately after z' and continuing all the way to v itself. However, since z' is minimal, there are no nodes on permitted paths from x to z' with direct links to w; so we can see using the inductive hypothesis that z' will not be marked with M_{pre}. Therefore, w will be marked with M_{tt} in lines 9–12 of *check-preemption(c)*, but not with M_{ff} in lines 11–12; and so on line 13, w will be marked with M_T.

Part 2: soundness for positive paths. Suppose that *query(x,y)* marks w with M_T on the n-th iteration of the main loop. There are only three commands in the loop that could result in w's being marked with M_T: (i) the command in line 6 of *query(x,y)*, (ii) the command in line 12 of *query(x,y)*, and (iii) the command in line 13 of *check-preemption(c)*. We examine them in turn, showing that no matter how M_T is assigned to w, it must turn out that Γ supports $x \rightarrow w$.

(i) If w is marked with M_T in line 6 of *query(x,y)*, it must be that $x \rightarrow w \in \Gamma$. Hence, Γ supports $x \rightarrow w$.

(ii) Suppose w is marked with M_T in line 12 of *query(x,y)*. Then w must have been marked with M_{tt} in lines 8–9. Looking at lines 8–9, we can see therefore, that there must be some node u with $u \rightarrow w \in \Gamma$ such that *query(x,y)* marks u with M_T. By inductive hypothesis, then, $\Gamma \models x \rightarrow \sigma_1 \rightarrow u$, for some path σ_1. This tells us that clauses II.1.(a) and II.1.(b) of the inheritance definition are satisfied. Moreover, II.1.(c) is satisfied as well: we know that $x \not\rightarrow w \notin \Gamma$—since if $x \rightarrow w \in \Gamma$, then w would have been marked with M_F at line 7, and so w could not have been marked with M_{tt} in lines 8–9. Finally, suppose there were some node v such that $\Gamma \models x \rightarrow \tau \rightarrow v$ and $v \not\rightarrow y \in \Gamma$. By inductive hypothesis, *query(x,y)* would already have marked v with M_T on a previous iteration of the loop. The node w would then be marked with M_{ff} in lines 10–11 of the current iteration, and so w

could not be marked with M_T at line 12, contrary to our supposition. Hence, there can be no such node v, and II.1.(d) is vacuously true. Therefore, since the clauses II.1.(a) through II.1.(d) are satisfied, $\Gamma \models x \rightarrow \sigma_1 \rightarrow u \rightarrow w$, and so Γ supports $x \rightarrow w$.

(iii) Suppose w is marked with M_T in line 13 of *check-preemption(c)*, with $c = w$. In order for *check-preemption(c)* even to be called on the node w, the node must have been marked with M_{tt} in lines 8–9 of *query(x,y)*, and also with M_{ff} in lines 10–11. Since w is marked with M_{tt}, we know by an argument identical to that presented in (ii) that $\Gamma \models x \rightarrow \sigma_1 \rightarrow u$, that $u \rightarrow w \in \Gamma$, and that $x \not\rightarrow w \notin \Gamma$. So clauses II.1.(a) through II.1.(c) of the inheritance definition are satisfied. Since w is also marked with M_{ff}, however, an analogous argument tells us that there exist nodes v such that $\Gamma \models x \rightarrow \tau \rightarrow v$ and $v \not\rightarrow y \in \Gamma$.

If II.1.(d) were false, there would be some such node v for which there is no node z such that $\Gamma \models x \rightarrow \tau_1 \rightarrow z \rightarrow \tau_2 \rightarrow v$ and $z \rightarrow w \in \Gamma$. Let v' be a node satisfying these conditions of minimal degree^{x,y}. Since w is marked with M_T on line 13 of *check-preemption(c)*, it cannot have been marked with M_{ff} in lines 11–12; therefore, v' must have been marked with M_{pre} by the loop in lines 5–8. The effect of this loop, however, is to propagate the marker M_{pre} up a positive path, all of whose nodes are marked with M_T but not with M_F, from a node marked with M_T and linked directly to w. Therefore, we know that by the n-th iteration of the main loop of *query(x,y)* there must exist a sequence of nodes $z_0, z_1, \ldots, z_m$ occurring in Γ such that: each z_i is marked with M_T, none of the z_i's except perhaps z_0 is marked with M_F, $z_0 \rightarrow w \in \Gamma$, $z_m = v'$, and $z_i \rightarrow z_{i+1} \in \Gamma$ for all $0 \leq i < m$. The inductive hypothesis tells us that, for $0 \leq i \leq m$, Γ supports each of the statements $x \rightarrow z_i$, and also that, for $0 < i \leq m$, none of the statements $x \not\rightarrow z_i$ is contained in Γ. We can then conclude from Lemma 5 that Γ permits a path of the form $x \rightarrow \tau_0 \rightarrow z_0 \rightarrow z_1 \rightarrow \cdots \rightarrow z_n (= v')$. Since $z_0 \rightarrow w \in \Gamma$, this contradicts the above assumption concerning v'; and so II.1.(d) must be true. Γ therefore permits the path $x \rightarrow \sigma_1 \rightarrow u$ and so supports $x \rightarrow w$.

Part 3: completeness for negative paths. Similar to Part 1.

Part 4 : soundness for negative paths. Similar to Part 2. ∎

Lemma 5 *For a sequence of nodes $z_0, z_1, \ldots, z_m$, suppose: (i) that Γ supports $x \rightarrow z_i$ for $0 \leq i \leq m$, (ii) that $z_i \rightarrow z_{i+1} \in \Gamma$ for $0 \leq i < m$, and (iii) that $x \not\rightarrow z_i \notin \Gamma$ for $0 < i \leq m$. Then Γ permits a path of the form $x \rightarrow \tau_0 \rightarrow z_0 \rightarrow z_1 \rightarrow \cdots \rightarrow z_m$.*

Proof. It follows from (i) that $\Gamma \models x \rightarrow \tau_0 \rightarrow z_0$, for some τ_0. Assuming $\Gamma \models x \rightarrow \tau_0 \rightarrow z_0 \rightarrow z_1 \rightarrow \cdots \rightarrow z_n$, for some $n < m$, we show that $\Gamma \models x \rightarrow \tau_0 \rightarrow z_0 \rightarrow z_1 \rightarrow \cdots \rightarrow z_n \rightarrow z_{n+1}$. We satisfy II.1.(a) by inductive hypothesis; we satisfy II.1.(b) by (ii) and II.1.(c) by (iii). Finally, suppose there exists v such that $\Gamma \models x \rightarrow \tau \rightarrow v$ with $v \not\rightarrow z_{n+1} \in \Gamma$. By (i) we know that $\Gamma \models x \rightarrow \tau_{n+1} \rightarrow z_{n+1}$ for some path τ_{n+1}. Therefore there exists z such that $z \rightarrow z_{n+1} \in \Gamma$ and either $z = x$ or $\Gamma \models x \rightarrow \tau' \rightarrow z \rightarrow \tau'' \rightarrow v$. So II.1.(d) is satisfied as well. ∎

6.4 Performance of the algorithm

We now give bounds on the running time of *query(x,y)* in a net Γ.

Let $d(x)$ be the length of the longest shortest path from x to any other node in Γ. The procedure *trim-for-query(x,y)*, which propagates M_1 up and M_2 down the hierarchy between x and y, runs in $O(d(x))$ time. Note that $d(x)$ depends on the network as a whole; it does not depend on y. In most practical applications, however, $d(x)$ is not expected to be significant.[5]

The procedure *select-next-degree()* contains no loops. It therefore runs in constant time.

Let $C(x,y)$ represent the number of conflicted nodes encountered when computing *query(x,y)* in Γ. Conflicted nodes are terminal nodes of compound paths that conflict with some other compound path, but not with any direct path. That is, the conflicted nodes in Γ are those nodes c such that: (i) Γ permits paths of the form $x \rightarrow \sigma \rightarrow u$ and $x \rightarrow \tau \rightarrow v$, where Γ contains the both the links $u \rightarrow y$ and $v \not\rightarrow y$, and (ii) Γ contains neither of the links $x \rightarrow y$ or $x \not\rightarrow y$. These nodes will be marked with both M_{tt} and M_{ff} by *query(x,y)*; then *check-preemption* must be called sequentially for each such node. Let $d(x,y)$ denote the length of the longest shortest path from x to any other node in the trimmed network $\Gamma^{x,y}$. Obviously, $d(x,y) \leq d(x)$. The procedure *check-preemption* contains a single loop that propagates M_{pre} up "$\rightarrow$" links in $\Gamma^{x,y}$; it runs in $O(d(x,y))$ time.

[5] If inheritance were completely stable (see Section 5.3), the running time for *trim-for-query(x,y)* could be incrementally reduced to constant time, or even a single time step, by modifying the query procedure so that it adds redundant links to Γ as a side effect until $d(x)$ falls below the desired value. Since inheritance fails to exhibit generic stability, however, we would have to resort instead to more complex *conditioning* procedures (mentioned below, in Section 6.5) to improve the running time of this part of the algorithm.

Let $D(x,y)$ be the "depth" of the trimmed network $\Gamma^{x,y}$—that is, the degree^{x,y} of the node y. Note that $d(x,y) \leq D(x,y)$. The main loop of *query(x,y)* is executed at most $1 + D(x,y)$ times. Inside this loop is the **for** loop that calls *check-preemption(c)*, the total number of calls to which is $C(x,y)$, each running in time $O(d(x,y))$. Therefore, the running time of the entire procedure *query(x,y)* is

$$O(max(d(x), D(x,y) + [C(x,y) \cdot d(x,y)])).$$

Since $d(x)$ is not expected to be a significant factor in realistic knowledge bases, the running time of a *query(x,y)* in such a knowledge base will be

$$O(D(x,y) + [C(x,y) \cdot d(x,y)]).$$

In a realistic knowledge base containing very few conflicted nodes, the running time of *query(x,y)* will of course approach the purely parallel $O(D(x,y))$.

6.5 Discussion

The PMPM architecture we described in this section is rather limited in computational power, which makes economical implementations possible. Fahlman estimated in [6] that a million-element marker propagation machine could be constructed using just a few custom VLSI chips, plus a lot of RAM. The Connection Machine [8], which is the closest physical realization to our idealized PMPM, was inspired by Fahlman's work; the algorithm we present here is thus well suited to the Connection Machine. Our algorithm also maps very naturally onto other parallel architectures. We briefly mention two extensions.

First, suppose the PMPM architecture is modified so that, in a network with N nodes, each node has at least $2N$ marker bits, so that the entire theory of the network can be represented at once. We reserve the bits M_{T_x} and M_{F_x} for each node x in the net. If y is marked with M_{T_x} it indicates that Γ permits $x \rightarrow y$, and similarly for M_{F_x}. A PMPM with this many marker bits could simply compute the entire theory of a network at once, and then use table lookup to answer queries. A more practical approach would be to compute the results of queries as needed and cache them in the nodes.

A second possible extension involves the sequential calls to check-preemption(c) in the body of the **for** loop in the main inference procedure. Suppose the we had the ability to propagate m different sets of markers independently, in parallel. (This might be the case if a PMPM were being simulated on a dataflow architecture with m processors.) We

could then create m sets of markers $M_{c,i}$, $M_{tt,i}$, $M_{ff,i}$, $M_{dir,i}$, and $M_{pre,i}$, for $1 \leq i \leq m$, and process conflicted nodes of degree i in parallel, m at a time.

Two related inheritance reasoners that have appeared in the literature recently are Touretzky's TINA [16], and Etherington's nondeterministic algorithm [4]. Both are implementations of credulous rather than skeptical definitions. TINA (for Topological Inference Architecture) computes the credulous extension of a network according to the theory provided by [16], provided that the extension is unique. If the network has multiple credulous extensions, TINA detects this fact and issues an error message. Etherington's algorithm generates one credulous extension, but which one depends on the choice of order in which markers are propagated; this is the source of the nondeterminism. Etherington notes that in some cases the algorithm will never choose certain extensions which his theory permits.

In addition to computing the extension, TINA had another function. In a process known as "conditioning," it augmented the inheritance network with extra links so that a marker propagation algorithm, called an *upscan*, could reconstruct portions of the extension as needed. The upscan algorithm was quite simple: it used shortest-path reasoning and had no sequential bottleneck, unlike the algorithm presented here. We should emphasize, however, that its purpose was to reconstruct an extension rather than compute it.[6] The conditioning process, which takes a network and its correct extension as inputs, alters the network as necessary to ensure that upscans produce correct results. A similar trick could be used with any inheritance definition.

TINA also supported a second kind of scan, called a downscan. A downscan of y marked all nodes x such that Γ permits $x \rightarrow y$ or (represented by a different marker) $x \not\rightarrow y$. Downscans are useful for finding all the members of a set, which can then be intersected with other sets in parallel—for example, to find all the gray elephants by intersecting elephants and gray things. It would appear to be far more expensive to compute downscans than upscans on a PMPM. Each conflicted node c encountered during a downscan from y would have to be resolved individually by calling *query*(c, y).

[6]It does not appear possible to compute extensions on a PMPM according to the inheritance definition provided in [16], even when the extension is unique. The difference appears to be unrelated to the choice between the skeptical and credulous approaches. Instead, it derives from the fact that [16] relies on a slightly different treatment of preemption, which forces a reasoning architecture to pay more attention to actual paths, rather than just supported conclusions. We compare these two styles of preemption in [17].

7 Conclusion

We have presented in this paper a new, skeptical theory of inheritance reasoning in nonmonotonic semantic networks. As far as we know, this theory represents the first significant alternative to the analysis of nonmonotonic inheritance reasoning presented in [16]. (A less radical alternative is described by Sandewall in [14]; although it differs in some ways from Touretzky's, Sandewall's is nevertheless a credulous theory.) The fact that there should be distinct but, perhaps, equally well-motivated accounts of correct reasoning in this context comes as something of a surprise; it is reminiscent of the situation in philosophical logic, where there exist rival logics embodying distinct conceptions of correct deductive reasoning.

In the context of inheritance reasoning, the existence of these distinct approaches raises a number of issues, which we are exploring in our current research. Much of this research is focused more or less directly on inheritance theory: we are studying the relations among the different analyses of nonmonotonic inheritance reasoning [17], and working to extend some of these analyses to more expressive nonmonotonic network languages.

However, it is also possible that this research will shed some light on more general treatments of nonmonotonic reasoning. It has been shown by Etherington [4], for example, that the default logic of Reiter [12] can be used to provide a specification for correct inheritance reasoning in nonmonotonic semantic networks: Etherington establishes a close correspondence between these networks and certain kinds of default theories ("network default theories"). But these results, linking default logic to nonmonotonic inheritance, presuppose a *credulous* analysis of inheritance reasoning; this bias towards a credulous approach to nonmonotonic reasoning is in fact built into both Reiter's default logic and the nonmonotonic logic of McDermott and Doyle [10]. Since, as we have shown, there turns out to be an equally well-motivated *skeptical* theory of nonmonotonic reasoning, at least in the case of semantic networks, it might be useful at this point to seek a weaker version of default or nonmonotonic logic, exhibiting instead a bias toward skepticism—or perhaps a more general logic that is neutral between the credulous and skeptical approaches.

A A skeptical reasoner in Common Lisp

This appendix presents a direct Common Lisp implementation of the inheritance definition in Section 4. Unlike the PMPM algorithm of Section 6, this direct implementation computes the extension of a net, by sequentially enumerating all permitted paths. Since the number of paths in an extension can grow exponentially with the number of nodes, this approach is not practical for large networks.

The algorithm is a line-by-line translation of the definition in Section 4, except that, for reasons of efficiency, a different measure of degree is used. Rather than evaluate paths in order of increasing degree, the Lisp program performs a topological sort on the entire graph and orders paths according to the number $T(z)$ which the topological sort assigned to the last node z of each path. Thus, if Γ consists of the links $x \rightarrow y$ and $y \rightarrow z$, $T(z) = 2$, so all paths ending with z have degree 2. In contrast, since $\deg_\Gamma$ is based on path length, $\deg_\Gamma(y \rightarrow z) = 1$.

It is easily shown that if $\sigma_1 = x \rightarrow \tau_1 \rightarrow y_1$ and $\sigma_2 = x \rightarrow \tau_2 \rightarrow y_2$, then $T(y_1) < T(y_2)$ iff $\deg_\Gamma(\sigma_1) < \deg_\Gamma(\sigma_2)$. Therefore, a program whose notion of path complexity is based on topological order will always produce results in agreement with the other measures of degree we have defined. The program may consider paths in a slightly different order than if it were using $\deg_\Gamma$, but it will only do so in situations where this does not affect the result.

The code for our Common Lisp reasoner is contained in Appendix A.1, beginning on the following page. The main inference function is `theory-of`. It uses the subfunctions `admissible-pos?` to check clauses II.1.(c) and II.1.(d) of the inheritance definition and `admissible-neg?` to check clauses II.2.(c) and II.2.(d). The top level functions are `load-net`, `show-theory`, and `show-theory-of`. Two sample input files are shown in Appendix A.2, and sample runs in Appendix A.3.

A.1 The Common Lisp code

```
;;; -*- Base: 10; Mode: LISP; Package: USER; Syntax: Common-lisp -*-

;;; A skeptical inheritance reasoner.


(defstruct (node (:print-function print-node))
  name
  (level 0)
  (pos-links nil)
  (neg-links nil))


(defstruct (link (:print-function print-link))
  tail
  arrow
  head)


(defconstant arrow-types '(---> -/->))


(defvar *the-net*   nil "List of linkspecs (TAIL ARROW HEAD).")
(defvar *nodelist*  nil "Nodes referenced in the network.")
(defvar *linklist*  nil "Links referenced in the network.")
(defvar *max-depth* 0   "Depth of the IS-A subgraph.")



(defun initialize-reasoner ()
  (setq *the-net* nil)
  (setq *nodelist* nil)
  (setq *linklist* nil)
  (setq *max-depth* 0))
```

```
;;; Code to load a network description.

(defun load-net (linkspecs)
  "Load a fresh network."
  (initialize-reasoner)
  (setq *the-net* linkspecs)
  (mapc #'add-link linkspecs)
  (topsort)
  (setq *max-depth* (node-level (car (last *nodelist*))))
  (mapcar #'node-name *nodelist*))


(defun add-node (x)
  (or (find x *nodelist* :key #'node-name)
      (car (push (make-node :name x) *nodelist*))))


(defun add-link (linkspec)
  "Adds one link of form (TAIL ARROW HEAD) to the network."
  (unless (and (listp linkspec)
               (= (length linkspec) 3.)
               (symbolp (first linkspec))
               (member (second linkspec) arrow-types)
               (symbolp (third linkspec)))
      (error "~S is an invalid link specfication" linkspec))
  (let* ((tail  (add-node (first linkspec)))
         (arrow (second linkspec))
         (head  (add-node (third linkspec)))
         (link  (make-link :head head :tail tail :arrow arrow)))
    (push link *linklist*)
    (case arrow
      (---> (push link (node-pos-links tail)))
      (-/-> (push link (node-neg-links tail))))
    link))



;;; Topological sort of the IS-A subgraph.
;;; Computes the NODE-LEVEL field of all nodes in *NODELIST*.
;;; Finally, sorts *NODELIST*.

(defun topsort ()
  "Topologically sort the nodes of the inheritance graph."
  (do ((len (length *nodelist*))
       (change-flag nil nil)
       (another-pass? t change-flag)
       (i 0. (if (< i len) (1+ i)
                 (error "Network is circular!!!"))))
      ((not another-pass?))
    (dolist (link *linklist*)
      (when (<= (node-level (link-head link))
                (node-level (link-tail link)))
        (setf (node-level (link-head link))
              (1+ (node-level (link-tail link))))
        (setf change-flag t))))
  (setq *nodelist* (sort *nodelist* #'< :key #'node-level)))
```

```
;;; Generate the skeptical theory of a node.

(defmacro first-node (path)
  `(link-tail (car ,path)))

(defmacro last-node (path)
  `(link-head (car (last ,path))))

(defmacro extend (path link)
  `(append ,path (list ,link)))

(defmacro target? (node links)
  "Returns T if NODE is the head of any of LINKS."
  `(member ,node ,links :key #'link-head))



(defun theory-of-node (x)
  "Returns theory of node X as a pair (POS-PATHS NEG-PATHS)."

  (let ((pos-paths (mapcar #'list (node-pos-links x)))   ;;Case I
        (neg-paths (mapcar #'list (node-neg-links x))))

      ;;Case II
    (do ((current-level (+ 2 (node-level x)) (1+ current-level)))
        ((> current-level *max-depth*))

        ;;Clauses II.1.(a) and II.2.(a)
      (dolist (a-path pos-paths)

          ;;Clause II.1.(b)
        (dolist (next-link (node-pos-links (last-node a-path)))
          (if (= (node-level (link-head next-link)) current-level)
              (let ((new-path (extend a-path next-link)))
                (if (admissible-pos? new-path pos-paths)
                    (push new-path pos-paths)))))

          ;;Clause II.2.(b)
        (dolist (next-link (node-neg-links (last-node a-path)))
          (if (= (node-level (link-head next-link)) current-level)
              (let ((new-path (extend a-path next-link)))
                (if (admissible-neg? new-path pos-paths)
                    (push new-path neg-paths)))))))

    (list (nreverse pos-paths) (nreverse neg-paths))))
```

```
(defun admissible-pos? (new-path earlier-pos-paths)
  "Returns T if NEW-PATH is admissible given EARLIER-POS-PATHS."
  (let ((x (first-node new-path))
        (y (last-node new-path)))
    (and   ;;Clause II.1.(c)
         (not (target? y (node-neg-links x)))
           ;;Clause II.1.(d)
         (every #'(lambda (path-1 &aux (v (last-node path-1)))
                    (if (target? y (node-neg-links v))
                        (find-pos-preemptor v y earlier-pos-paths)
                        t))
                earlier-pos-paths))))


(defun find-pos-preemptor (v y earlier-pos-paths)
  (some #'(lambda (path-2)
            (and (eq (last-node path-2) v)
                 (some #'(lambda (link-2 &aux (z (link-head link-2)))
                           (and (not (eq z v))
                                (target? y (node-pos-links z))))
                       path-2)))
        earlier-pos-paths))



(defun admissible-neg? (new-path earlier-pos-paths)
  "Returns T if negative NEW-PATH is admissible given EARLIER-POS-PATHS."
  (let ((x (first-node new-path))
        (y (last-node new-path)))
    (and   ;;Clause II.2.(c)
         (not (target? y (node-pos-links x)))
           ;;Clause II.2.(d)
         (every #'(lambda (path-1 &aux (v (last-node path-1)))
                    (if (target? y (node-pos-links v))
                        (find-neg-preemptor v y earlier-pos-paths)
                        t))
                earlier-pos-paths))))


(defun find-neg-preemptor (v y earlier-pos-paths)
  (some #'(lambda (path-2)
            (and (eq (last-node path-2) v)
                 (some #'(lambda (link-2 &aux (z (link-head link-2)))
                           (and (not (eq z v))
                                (target? y (node-neg-links z))))
                       path-2)))
        earlier-pos-paths))
```

```
;;; Display theories and inference paths.

(defun show-theory ()
  "Computes and displays the theory of the currently loaded network."
  (mapc #'show-theory-of (reverse *nodelist*))
  nil)


(defun show-theory-of (node)
  "Computes and displays the theory of NODE."
  (let* ((x (cond ((node-p node) node)
                  ((find node *nodelist* :key #'node-name))
                  (t (return-from show-theory-of-node nil))))
         (th (theory-of-node x)))
    (mapc #'show-path (car th))
    (mapc #'show-path (cadr th))
    nil))


(defun show-path (path)
  (format t "~%~S" (node-name (first-node path)))
  (dolist (link path)
    (format t " ~S ~S" (link-arrow link) (node-name (link-head link)))))




;;; Miscellaneous functions.

(defun print-node (node stream depth)
  (declare (ignore depth))
  (format stream "#<Node ~S>" (node-name node)))


(defun print-link (link stream depth)
  (declare (ignore depth))
  (format stream "#<Link ~S ~S ~S>"
          (node-name (link-tail link))
          (link-arrow link)
          (node-name (link-head link))))
```

A.2 Sample input files

```
;;; File:  Preemption-Example.Lisp
;;;
;;; Demonstrates preemption in the presence of a redundant link.

(load-net '(
  (clyde ---> royal.elephant)
  (clyde ---> elephant)
  (royal.elephant ---> elephant)
  (elephant ---> gray.thing)
  (royal.elephant -/-> gray.thing)))
```

```
;;; File:  Nixon-Diamond.Lisp
;;;
;;; The canonical ambiguous network.

(load-net '(
  (nixon ---> republican)
  (nixon ---> quaker)
  (quaker ---> pacifist)
  (republican -/-> pacifist)))
```

A.3 Sample runs

```
(load "Preemption-Example.Lisp")
(CLYDE ROYAL.ELEPHANT ELEPHANT GRAY.THING)
```

```
(show-theory-of 'clyde)
CLYDE ---> ROYAL.ELEPHANT
CLYDE ---> ELEPHANT
CLYDE ---> ROYAL.ELEPHANT ---> ELEPHANT
CLYDE ---> ROYAL.ELEPHANT -/-> GRAY.THING
NIL
```

```
(show-theory)
ELEPHANT ---> GRAY.THING
ROYAL-ELEPHANT ---> ELEPHANT
ROYAL-ELEPHANT -/-> GRAY.THING
CLYDE ---> ROYAL.ELEPHANT
CLYDE ---> ELEPHANT
CLYDE ---> ROYAL.ELEPHANT ---> ELEPHANT
CLYDE ---> ROYAL.ELEPHANT -/-> GRAY.THING
NIL
```

```
(load "Nixon-Diamond.Lisp")
(QUAKER PACIFIST REPUBLICAN NIXON)


(show-theory)
QUAKER ---> PACIFIST
REPUBLICAN -/-> PACIFIST
NIXON ---> QUAKER
NIXON ---> REPUBLICAN
NIL
```

REFERENCES

[1] N. Belnap. How a computer should think. In G. Ryle (ed.), *Contemporary Aspects of Philosophy.* Oriel Press (1977), pp. 30–56.

[2] N. Belnap. A useful four-valued logic. In J. Dunn and G. Epstein (eds.), *Modern Uses of Multiple-valued Logic.* D. Reidel (1977), pp. 8–37.

[3] G. Carlson. Generic terms and generic sentences. *Journal of Philosophical Logic*, vol. 11 (1982), pp. 145-181.

[4] D. Etherington. Formalizing nonmonotonic reasoning systems. *Artificial Intelligence*, vol. 31 (1987), pp. 41-85.

[5] S. Fahlman. *NETL: a System for Representing and Using Real-world Knowledge.* The MIT Press (1979).

[6] S. Fahlman. Design sketch for a million-element NETL machine. *Proceedings of AAAI-80* (1980), pp. 249–252.

[7] S. Fahlman, D. Touretzky, and W. van Roggen. Cancellation in a parallel semantic network. *Proceedings of IJCAI-81* (1981), pp. 257–263.

[8] W. D. Hillis. *The Connection Machine.* The MIT Press (1985).

[9] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, vol. 13 (1980), pp. 27–39.

[10] D. McDermott and J. Doyle. Non-monotonic logic I. *Artificial Intelligence*, vol. 13 (1980), pp. 41–72.

[11] M. Quillian. Semantic memory. Ph.D. Dissertation, Carnegie Institute of Technology (1966). Reprinted in M. Minsky (ed.), *Semantic Information Processing*, The MIT Press (1968).

[12] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, vol. 13 (1980), pp. 81–132.

[13] R. Roberts and I. Goldstein. *The FRL Manual.* AI Memo No. 409, MIT Artificial Intelligence Laboratory (1977).

[14] E. Sandewall. Non-monotonic inference rules for multiple inheritance with exceptions. *Proceedings of the IEEE*, vol. 74 (1986), pp. 1345–1353.

[15] R. Thomason, J. Horty, and D. Touretzky. A calculus for inheritance in monotonic semantic nets. Technical Report CMU-CS-86-138, Computer Science Department, Carnegie Mellon University (1986).

[16] D. Touretzky. *The Mathematics of Inheritance Systems*. Morgan Kaufmann (1986).

[17] D. Touretzky, J. Horty, and R. Thomason. A clash of intuitions: the current state of nonmonotonic multiple inheritance systems. In *Proceedings of IJCAI-87* (1987).